UCRL-53921 th

An Investigation of the Usefulness of Earley's Algorithm in a General Context-Free Recognizer

Cathleen M. Benedetti

(M.S. Thesis)

June 1989

Lawrence Livermore National Laboratory

CIRCULATION COPY
SUBJECT TO RECALL
IN TWO WEEKS

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (615) 576-8401, FTS 626-8401

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	001–050
A03	051–100
A04	101–200
A05	201–300
A06	301–400
A07	401–500
A08	501–600
A09	601

UCRL-53921 th
Distribution Category UC-705

An Investigation of the Usefulness of Earley's Algorithm in a General Context-Free Recognizer

Cathleen M. Benedetti

(M.S. Thesis)

Manuscript date: June 1989

LAWRENCE LIVERMORE NATIONAL LABORATORY
University of California • Livermore, California • 94551

Available to DOE and DOE contractors from the Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831 Prices available from (615) 576-8601, FTS 626-8401

Available to the public from the National Technical Information Service, U.S. Department of Commerce
5285 Port Royal Rd., Springfield, VA 22161 • • (Microfiche A01)

An Investigation of the Usefulness of Earley's Algorithm in a General Context-Free Recognizer

By

Cathleen M. Benedetti

THESIS

Submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Meera M. Blattner

Paul F. Dubois

[illegible]

Committee in Charge

Abstract

This thesis explores the potential applications of a general context-free recognizer implemented using Earley's Algorithm. First, several potential applications are identified. The recognizer is implemented, and tested using several grammars and input strings representative of these applications. The results of these tests are then examined to determine whether Earley's Algorithm can be successfully used in a general context-free recognizer.

TABLE OF CONTENTS

1. OVERVIEW........ 1
2. EARLEY'S ALGORITHM........ 4
3. APPLICATIONS OF EARLEY'S ALGORITHM........ 7
4. OBJECT-ORIENTED LANGUAGES........ 10
5. EIFFEL........ 11
6. IMPLEMENTATION........ 12
7. TEST CASES........ 19
8. RESULTS........ 21
9. DISCUSSION........ 34

REFERENCES........ 36

APPENDIX A: Earley's Algorithm........ 38

APPENDIX B: The Recognizer........ 40

APPENDIX C: Input Grammars........ 57

APPENDIX D: Input Strings........ 85

OVERVIEW

Earley's algorithm is a general context-free recognition algorithm proposed by Jay Earley in 1968 [9]. Given any context-free grammar and an input string, the algorithm determines whether the input string is a member of the language described by the grammar. Since the introduction of Earley's algorithm into the literature, little has been done to test Earley's original conclusions about the complexities of the algorithm. The complexity of Earley's algorithm is determined by the total number of states (see Appendix A) which can be created when the algorithm is given a particular grammar and an input string. The complexities which Earley gives in [9] are given in terms of n, the length of the input string. Earley's algorithm is known to be of complexity $O(n^3)$ for arbitrary context-free grammars, and $O(n^2)$ for unambiguous context-free grammars. Its complexity is O(n) for a class of grammars that includes LR(k)[1] grammars [9]. The algorithm also has the advantage that it does not require the input grammar to be in any special form.

While Earley's algorithm has been modified for use in many applications, his original conclusion that the algorithm has complexity O(n) - $O(n^2)$ for unambiguous context-free grammars has not been documented in the literature. Furthermore, these general orders alone give little indication of whether the algorithm is of any practical use. To determine this, it is necessary to implement the algorithm and record its execution time for several different grammars and input strings. These grammars and input strings represent situations and applications where it would be desirable to use a general context-free recognizer. By observing the actual behavior of Earley's algorithm on real grammars, we can better judge its potential to be successfully used in software applications.

The reason for this investigation is to examine the usefulness of incorporating Earley's algorithm into a general context-free recognizer or parser. Of particular interest is

---

[1] LR(k) grammars are those which are recognized by parsers that use LR(k) parsing algorithms. Aho, Sethi, & Ullman [2] explain that "the technique is called LR(k) parsing; the 'L' is for left-to right scanning of the input, the 'R' for constructing a rightmost derivation in reverse, and the k for the number of input symbols of lookahead that are used in making parsing decisions. When (k) is omitted, k is assumed to be 1" (p. 215). A complete description of LR(k) parsing can be found in [2].

the use of these tools in situations where an LR parser or other method cannot be used, or where it is more desirable to use a general context-free recognizer, particularly one which is implemented using Early's algorithm. For example, often there are some desirable non-LR(1) constructions that could be added to a programming language that would be more natural to use than similar constructions in LR(1) grammars. The addition of such non-LR(1) expressions gives more flexibility and power to an LR(1) programming language, and allows for greater ease of use. Earley's algorithm offers the hope of efficient implementations of a recognizer, parser, and compiler-compiler for these modified programming languages. Such a compiler-compiler allows language designers the flexibility to use general context-free structures, rather than restrict them to smaller classes of grammars such as LR or LALR [2] grammars [2]. Tools which use Early's algorithm would be helpful in several other situations and applications. These include the beginning stages of compiler development, the checking of command files for errors, a multi-language parser, and the implementation of a variety of programming tools.

The first part of this thesis is the construction of a general context-free recognizer implemented with Earley's algorithm. This recognizer is used to help determine whether it is feasible to incorporate such a recognizer into any of the previously mentioned applications. The recognizer accepts a context-free grammar and a string as input. Using Earley's algorithm, it determines whether or not the string is in the language described by the context-free grammar. The recognizer is implemented in Eiffel [11], an interesting new object-oriented programming language designed to promote the creation of reliable and reusable software. Eiffel was chosen because it offers a unique combination of programming language features, and, since it is relatively new, to examine how it fares as a programming environment.

The feasibility of using such a recognizer is determined by measuring its execution time and space requirements on existing and example grammars. These measurements are taken using several different input strings with each grammar. In this manner we can

---

[2] LALR grammars are those which are recognized by LALR parsers, a particular type of LR parser. The LA in LALR stands for look-ahead.

observe how the performance of Earley's algorithm varies with the length of the input string for a given grammar. It is also interesting to see how the performance of the recognizer varies from grammar to grammar with the same size input string. The grammars and input strings used in these tests are representative of situations and applications in which it would be desireable to use a general context-free recognizer, particularly one that uses Early's algorithm in its implementation.

These measurements can be used in several different ways. In this investigation, they will be used to identify the constants of the complexities of Early's algorithm for individual grammars. They will also show in what instances it is reasonable to use Earley's algorithm in a tool which accepts general context-free grammars, as well as whether it is practical to use that tool to expand existing programming language grammars.

EARLEY'S ALGORITHM

Earley's algorithm (see Appendix A, [9] and [10]) was chosen as part of the implementation of the recognizer because it is a general context-free recognition algorithm which appears to have reasonable time and space bounds for certain subsets of context-free grammars. In particular, the promise of a time complexity of O(n) for LR(k) grammars suggests that its use in expanding an LR(1) grammar (described below) would provide reasonable results. Earley's algorithm has two distinct advantages over another well-known context-free recognition algorithm, the Cocke-Younger-Kasami (CYK) algorithm [6, p. 302]. The CYK algorithm has a time complexity of $O(n^3)$ for all context-free grammars. Unlike the CYK algorithm, which requires its input grammars to be in Chomsky normal form[3], Earley's algorithm does not require its input grammars to be in any special form. Earley's algorithm is also known to have been successfully adapted for use in a commercially available syntax-directed editor.

Earley's algorithm has been modified for use in several applications and algorithms. Christopher, Hatcher & Kukuk [7] used Earley's algorithm to produce optimized code from intermediate code in two Graham-Glanville style code generators. Earley's algorithm was used to produce all possible parses. Values were assigned to the productions, and then the minimum cost parse was chosen. They also discuss the use of Earley's Algorithm in an experimental C code generator. Vol'dman [18] describes an algorithm for context-sensitive grammars which he describes as a "generalization of Earley's algorithm" (p. 302). In [5], Bouckaert, Pirotte and Snelling introduce a set of algorithms of which they claim Earley's algorithm to be a member. Some of these algorithms appear to do better than Earley's algorithm, but not in all instances. The interesting point which is made in [5] is that, while their new parsing algorithms lack efficiency when generally compared to parsers which restrict their input grammars, they seem to perform "fairly well on LL or LR grammars, while still being able to parse any CF-G" (p. 16). Chiang and Fu [6] modified Earley's Algorithm to a parallel recognition algorithm for use in VLSI implementations, restricting the

---

[3] A grammar written in Chomsky normal form contains only production rules of the form A ::= a or A ::= BC. For every context-free grammar, there is such an equivalent grammar.

input grammars to be λ-free. Zaitman and Kholodenko [19] also modified the algorithm for use in a multilanguage programming system.

This thesis attempts to judge the usefulness of Earley's algorithm. In doing so, the first question which must be answered is the one posed by Aho, Hopcroft & Ullmann [1, p. 2]. "How should we judge the goodness of an algorithm?" Earley characterizes those grammars for which his algorithm can recognize input strings in O(n), O(n^2), and O(n^3) time [9]. There are those computer scientists who will argue that these complexities are all the information needed to determine whether an algorithm is "good" or not. Realistically, this is not the case. In most instances, including this one, more information is needed before it can be determined whether the Earley's algorithm is practical, efficient, and useful.

In determining the complexity of his algorithm, Earley uses some of the characteristics of the input grammar [9]. For a given grammar, there are a maximum number of states which can be created based solely on the length of the input string and the characteristics of the grammar. This is the worst case, and it provides a theroretical upper bounds on the complexity of the algorithm in a particular instance. These characteristics include the number of production rules, the maximum number of symbols in a production rule, and the number of terminals in the grammar. Since each of these quantities is constant with respect to the length of the input string, they are not evident when the complexity is discussed solely in terms of n. However, these "constants" are determined by another input to the algorithm, the grammar, and they have the potential to greatly effect the execution time of the algorithm. One of the goals of this investigation is to determine how severly the execution time of the recognizer can be affected by the grammar.

In determining whether a recognizer which uses Earley's algorithm is of any practical use, three steps must be taken:

1) Identify potential applications.
2) Test Earley's algorithm on examples of these potential applications.
3) Evaluate the results of these tests.

In evaluating the results for each particular application, the experimental constants associated with the complexity of Earley's algorithm for individual grammars will be identified. Subjective judgements based on the execution times of the recognizer will then be made to determine whether Earley's algorithm can be successfully used in a general context-free recognizer in any of the applications posed in this thesis.

APPLICATIONS OF EARLEY'S ALGORITHM

This section describes the applications in which it would be desirable to use a general context-free recognizer, particularly a recognizer which uses Earley's algorithm in its implementation. These applications fall into two categories. The first group of applications are those which involve the use of a programming language grammar. These are large applications such as a compiler-compiler and sytax-directed editor. The second group consists of smaller applications and software tools.

There are many compiler-compilers available to assist compiler writers with implementation. Most of the efficient compiler-compilers use algorithms which place restrictions on the type of input grammar that the compiler-compiler can accept. Commonly, the systems use LR, LALR and LL[4] parsing algorithms. Complete descriptions of these algorithms are contained in Aho, Sethi, and Ullman [2]. Examples of such systems include the LR System [15] used at Lawrence Livermore National Laboratory (LLNL). It uses an LR algorithm [2], and therefore requires input grammars to be LR. Likewise, the Unix utility Yacc [16] uses a LALR algorithm, and requires its input grammars to be LALR. These and other such systems are widely available and used extensively to assist in the construction of efficient compilers.

By forcing a compiler writer to conform their grammar to a specific subset of context-free grammars, a compiler-compiler's efficiency and time complexity are greatly improved. The reason for this is that instead of having to provide for all context-free grammars, it only needs to know how to process a particular subset. If the grammar being processed fails to meet certain criteria, it is rejected, and the compiler writer must modify the grammar until an acceptable alternative is found. While this process may be frustrating for inexperienced grammar writers, in most instances the ability of the compiler-compiler to accept only one type of grammar is more than sufficient. Insisting that the grammar is in a particular form generally increases the efficiency and lowers the time complexity of the

---

[4] As with LR parsing algorithms, Aho, Sethi, & Ullman [2] explain that the first 'L' in LL "stands for scanning the input from left to right, the second 'L' for producing a leftmost derivation" (p. 191).

algorithm used by the compiler-compiler writer. However, there are situations when a compiler writer might want a more flexible solution.

This investigation was originally prompted by the desire to allow some non-LR(1) constructs into an existing LR(1) programming language grammar. When designing or expanding a programming language grammar, it is necessary to keep the users in mind. It is frequently the case that the construct which a language designer wishes to offer to users does not fit easily into an existing LR(1) grammar. After alterations are made to make the construct LR(1), it is often the user who ultimately pays the price. The new construct given to the user is less natural and more complicated to use and understand than the construct that the language designer originally conceived. This leaves the language designer wishing for a compiler-compiler that would excuse slight deviations from an LR(1) grammar.

There are other instances when a user would prefer a general context-free recognizer or parser. In the early stages of compiler development, when a grammar is undergoing constant and considerable change, it would be helpful to have a general context-free system. This allows the compiler writer to concentrate on implementation issues rather than the problems of a non-conforming grammar. A multi-language system, such as a parser or a syntax-directed editor is an ideal place for the capabilities of a general context-free recognizer. In these applications, it is preferable to have a system which provide the users with as much freedom has possible.

A general context-free recognizer can also be used in smaller applications. In these smaller applications, it would be undesirable to use an LR(1) recognizer because the users of these tools are not necessarily programming language experts, and therefore would have difficulty writing LR(1) grammars. These applications include a tool to check command files for errors before leaving for the night. This would help avoid the frustrating possiblity of returning in the morning only to discover that a program had not run because of a typographical error. In this situation, it is possible to use a searching algorithm. However, it would be more desireable to use a general context-free recognizer for two reasons. First, a search method would only be suitable in instances where there are a very small number of possible commands and a small amount of input. Second, when using a search method, allowances for the many options which frequently accompany a command must be made either in the list of commands, or in the code which processes the commands. Including these allowances in a command list would result in a large command list not suitable for use with a search method. It would also be much easier to include these options in a grammar, than to write code to accommodate each one.

Finally, a general parser can be used to assist in the implemention of programming tools. Rather than having to write the code to parse the input, a tool writer could supply a short grammar and leave the parsing work to the parser. This would certainly simplify the job of the tool writer.

As discussed above, it would be desirable to have a general context-free recognizer for several reasons. In a compiler-compiler, it gives compiler writers more freedom in language design and facilitates compiler implementation by allowing them to alter the grammar only when the language calls for it, not when the compiler-compiler mandates it. The recognizer provides generality for users of multi-language systems. In smaller applications, it helps with error checking and in parsing input without requiring the expertise to write an LR(1) grammar.

OBJECT-ORIENTED LANGUAGES

In recent years, much attention has been paid to the development and use of object-oriented languages and programming techniques. Briefly described here are the characteristics of object-oriented languages and how they influence programming style.

In object-oriented programming, emphasis is placed on the objects or data structures being manipulated, rather than on the procedures which are operating on the data. As Meyer [14] observes, "the primary design issue is not what the system does, but what objects it does it to" (p. 63). Classes are used to define data structures and the operations which can be performed on those structures. An object refers to a particular instantiation of a class.

The interface between the user and an object is strictly controlled. The user is only allowed access to an object through specific operations. In this manner, the representation of the object is hidden from the user. For example, a user who had an object of class STACK would have access to two operations: PUSH and POP. However, the implementation details, such as whether the stack was implemented by a linked list or an array, would not be visible. This allows both the author and client of a class the freedom to change their code without interferring with the other's work.

Another important feature of object-oriented programming is inheritance. Inheritance allows one class to inherit some or all of its structure from one or more previously defined classes. Already existing classes can be combined and extended to produce new classes. This ability promotes application independent code and helps to eliminate the unnecessary rewriting of modules for each new application.

The primary advantage of object-oriented programming is reusability. Modularity, information hiding through interface control, and inheritance all contribute to create software which can be used over and over again in entirely different applications. Classes can be grouped and re-grouped into new classes, all of which continue to expand the resources of the programmer.

EIFFEL

The language chosen to implement the recognizer is Eiffel [11]. Eiffel is a recently developed object-oriented language. It was chosen because it offers a unique combination of programming language features, and, since it is relatively new, to see how it fares as a programming environment.

Eiffel classes contain features. These features may be pieces of data or routines to manipulate that data. In its current release, Eiffel supports multiple inheritance (inheritance from more than one class), and repeated inheritance (inheritance from the same class more than once). Its feature redefinition and renaming capabilities add to its qualities as an object-oriented language. More information about Eiffel can be found in [11], [12], [13], and [14].

The choice of an object-oriented language, and in particular Eiffel, assisted greatly in the implementation of Earley's algorithm. In dealing with grammars, each part of the grammar is treated as an object, and those objects combine to form the larger object of the grammar. Representing the states and state sets of Earley's algorithm was also aided by the choice of Eiffel as the implementation language. State information and the states themselves could be combined in a way which made processing the states as straightforward as possible. Further details on the objects used in the implementation of the recognizer are contained in the next section on implementation.

Eiffel inheritance and feature redefinition cabilities were found to be of particular usefulness. They made it simple to build new classes and rewrite some of the existing library features to be compatible with the new classes. Overall, Eiffel is a well-organized language that facilitates programming tasks.

IMPLEMENTATION

This section describes the implementation of a general context-free recognizer that uses Earley's algorithm. A brief introduction is followed by a description of the classes and objects used in the recognizer. The two parts of the recognizer, REC1 and REC2, are then described in detail.

The implementation of the recognizer is divided into two parts. The first part, REC1, is responsible for reading the grammar and processing it into its run-time structure. The structure is then stored in a file using one of Eiffel's library routines. The second part, REC2, is responsible for retrieving the grammar and then processing the input string to determine whether or not the string is valid. The purpose in separating the process into two parts is to allow a grammar to be read once, then its structure stored in a file. This file may then be retrieved repeatedly and used to process any number of input strings without having to reread the entire grammar.

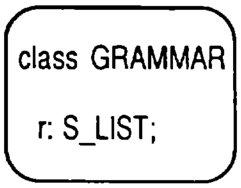
In the implementation of the recognizer, primarily two data structures, or objects, are needed. One object is used to represent the input grammar. The second object contains the representation of all the states and state sets of Earley's algorithm. The classes used to describe these objects will now be explained. To distinguish between classes and features, class names appear in uppercase type, while feature names appear in lowercase type. The figures in this section illustrate the inheritance structure and primary features of the classes. Appendix B contains the Eiffel code of the classes described here.

The class GRAMMAR (Figure 1) is used to represent the input grammar. Its most important feature is r, a sorted linked list of production rules represented by the class S_LIST (Figure 2). S_LIST inherits from the generic Eiffel library class LINKED_LIST [13]. The linked list is of class RULE (Figure 3), and is sorted by the RULE feature lhs (described below).

A RULE represents the information associated with a particular left hand side of a production rule. Its features include:

lhs:	a STRING [13] representing the left hand side of a production rule
rhs:	a linked list of SYMBOL_LIST (described below) representing all the alternatives of the left hand side
fst:	a SYMBOL_LIST representing the first set[5] of lhs

A SYMBOL_LIST (Figure 4) is a linked_list of STRING (an Eiffel library class), and like S_LIST inherits from the generic Eiffel library class LINKED_LIST.



```
class GRAMMAR

  r: S_LIST;
```

Figure 1

Class GRAMMAR

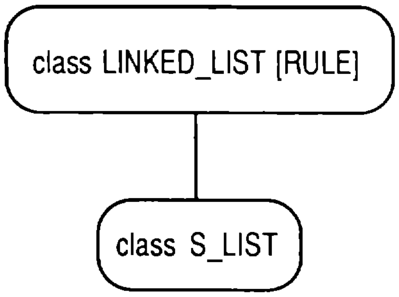
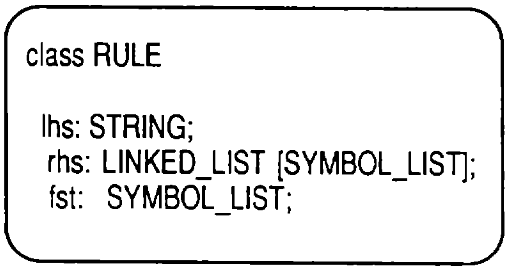


Figure 2

Class S_LIST



```
class RULE

 lhs: STRING;
  rhs: LINKED_LIST [SYMBOL_LIST];
  fst:  SYMBOL_LIST;
```

Figure 3

Class RULE

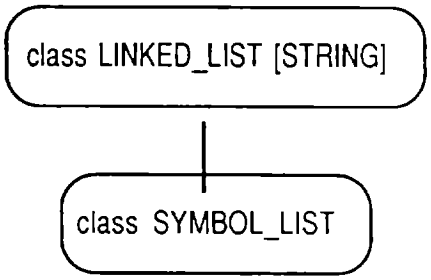


Figure 4

Class SYMBOL_LIST

---

[5] A first set is the set of all terminals which can begin a string derived from the left hand side of a production rule. A complete description of an algorithm for determining all of the first sets for a grammar can be found in Aho, Sethi & Ullman [2].

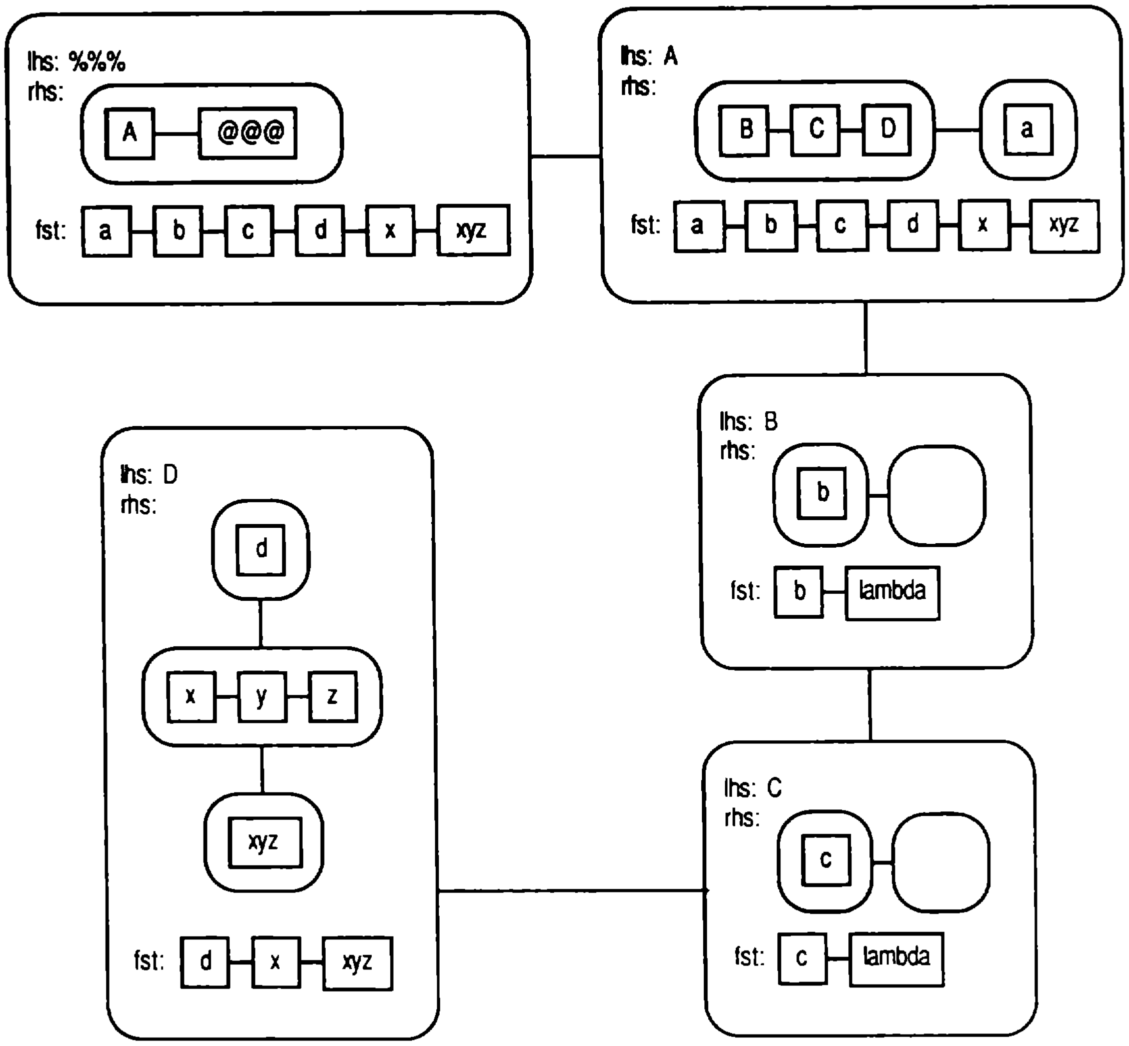


Figure 5

Example representation of a grammar

As an example of how a grammar is stored, consider the grammar:

A ::= B C D | a

B ::= b |

C ::= c |

D ::= d | x y z | xyz

Upper case letters indicate a non-terminal and lower case letters indicate terminals. In the recognizer, the grammar would be represented as is shown in Figure 5. The rules are stored alphabetically according to the left hand side. The first rule is added by the recognizer.

In Earley's algorithm, each state consists of a production rule, a position in the alternative of that rule, a look ahead string, and a pointer to a state set. The class STATE (Figure 6) is used to represent a state. Its features include:

lp:	an integer pointer to a position in the linked list which represents the grammar. It indicates the left hand side of the production.
rp:	an integer pointer to a position in the feature rhs of the RULE indicated by
lp.	indicates the production alternative.
dot:	an integer pointer to a position in the alternative.
look:	the look ahead string.
point:	an integer pointer to a STATE_SET

The class STATE_SET (Figure 7) joins these states together into a linked list by inheriting from LINKED_LIST. All the states are then contained in the feature als of REC2 (Figure 8), which is a linked list of STATE_SET.

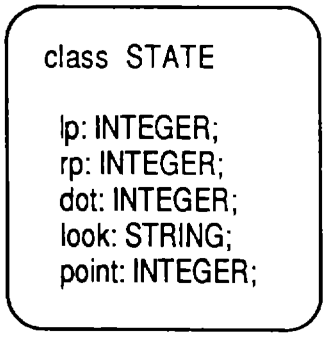


Figure 6

Class STATE

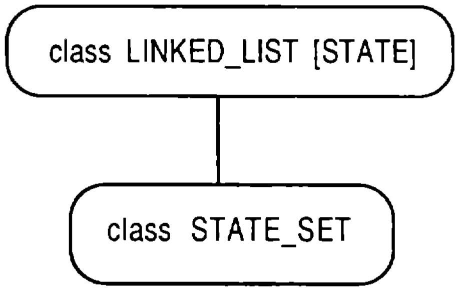


Figure 7

Class STATE_SET

```
class REC2

  als: LINKED_LIST [STATE_SET];
```

Figure 8

Class REC2

As mentioned above, the recognizer is divided into two parts. REC1, the first part of the recognizer, creates a GRAMMAR object, and then calls on the GRAMMAR feature READ_GR to read the input grammar from the file Z.BNF. READ_GR stores the rules in its feature R. If it does not encounter any errors, it continues on to find the first set of each left hand side and stores them in the feature R with the input grammar. After READ_GR is finished, REC1 stores the grammar structure in a file, GRAMMAR.STORE. Eiffel makes this task incredibly simple. The Eiffel library contains a class called STORABLE [13] Inheriting from this class allows a user to store entire objects with a single instruction. Objects can be retrieved just as easily. In this manner, the entire grammar structure can easily and quickly be stored, and then retrieved multiple times, eliminating the need for reprocessing the grammar.

REC2, the second part of the recognizer, picks up where REC1 left off. It retrieves the grammar structure from GRAMMAR.STORE, opens the file containing the input string, Z.INPUT, and proceeds to process the input string according to Earley's algorithm. Earley's algorithm will not be described in detail here. A short description can be found in Appendix A. More expansive descriptions can be found in [9] and [10].

While Earley's algorithm may seem straightforward at first glance, the implementation becomes more complicated when λ-rules (empty production rules) are allowed in the input grammar. Special allowances must be made in both the predictor and completer, as will now be described.

If the position of the dot in the current state is before a non-terminal, the predictor is used to operate on that state. Suppose the current state is:

```
S ::= . A + B      ;      1
```

and the alternatives of A are:

```
A ::= C + D
  ::= E + F
```

The predictor will add the following two states to the current state set:

```
A ::= . C + D     +     1
A ::= . E + F     +     1
```

In this case, the look-ahead is '+' because that is what follows A in the current state. If instead A was at the end of the production, then the look-ahead would be the look-ahead from the current state. However, when A is followed by another non-terminal, say X, a new state must be added for each terminal in the first set of X. This becomes complicated when one of the alternatives of X is a λ-rule. Then the process must be repeated until a non-terminal does not produce a λ-rule or the end of the production of the current state is reached.

Another example of λ-rules causing trouble is in the completer. The completer looks at the pointer of the current state, then examines the states in that state set. However, the pointer of a state which contains a λ-rule will be the current state set. This means that there are still more states which could be added to that state set. As part of the completer operating on a λ-rule state, these states must be examined. Since they are not yet known, an alternative solution must be found. In this implementation, a list is kept of those states which contain λ-rules, are in the current state set and have been processed by the completer. When a state that could potentially be affected is added to the current state set, it is examined. It is possible that this could result in more states being added which also need to be checked, and these states could add more states which also need to be checked and so forth.

Other than the above nuisances caused by λ-rules, the implementation of the recognizer was made relatively straightforward by the extensive use of linked lists, and the use of the object-oriented language Eiffel as the implementation language.

TEST CASES

Two groups of tests were conducted to measure the performance of Earley's algorithm. The first group of tests was designed to illustrate how Earley's algorithm performs in large applications involving programming language grammars. The second group of tests was designed to measure how Earley's algorithm performs in smaller applications involving small to moderate sized grammars and input strings. Several different grammars and input strings were used in timing the performance of the recognizer. Appendix C contains each of the grammars used in these tests. Appendix D contains the input strings.

To test the behavior of Earley's algorithm with programming language grammars as input, subsets of two existing programming languages were used. The first set of grammars describes subsets of an existing LR(1) programming language, Basis [8]. The second set of grammars describes subsets of Eiffel, the programming language used in the implementation of the recognizer. The original Eiffel grammar is contained in [12]. The grammars are named according to which set they belong to and the number of production rules they contain. For example, Basis and Eiffel grammars containingg 25 production rules would be named b.25 and e.25, respectively. The exceptions to this naming convention are the expression grammars b.aexp and b.lexp, which describe arithmetic and logical expressions, respectively.

The input strings used with the Basis and Eiffel grammar subsets are divided into groups according to which language constructs they contain. Members of different groups contain one or more different constructs. Like the grammars, the names of the input strings also begin with "b." and "e.", followed by their group name and the number of symbols they contain. In almost all cases, members of the same input string group are simply multiples of one another. The exceptions to this are the input strings for b.aexp and b.lexp (see Appendix D).

To test the behavior of Earley's algorithm with small, non-programming language grammars as input, two sets of grammars were used. The first set of input grammars describes computer commands. The second set of input grammars describe input to an existing programming tool used in conjuction with the Basis language [8], Config. As above, the grammars are named according to which set they belong to and the number of production rules they contain. The names of the Config grammars begin with "c" The names of the command (instruction) grammars begin with "i". Input strings are also named similarly. However, the input strings used in these smaller are grouped slightly differently since the input strings do not double in content when they double in size. Unlike the Basis and Eiffel input strings, input strings contained in the same group will contain different constructs. This is an important factor to remember when examining the execution times presented for these grammars and input strings in the following section.

The different grammars and input strings used in the timings of both parts of the recognizer, REC1 and REC2, will show not only how performance differs with differing input strings, but also how it varies depending on the size and nature of the input grammar. These results will be presented and discussed in the following sections, and will help determine the potentially successful uses for Earley's algorithm.

RESULTS

This section will present the results of using the test cases described in the previous section as input to the recognizer. The times given are shown in minutes and seconds and were recorded on a Sun IV Workstation. The error range is plus or minus .4 seconds. The results of the first group of tests involving the programming language grammars will be presented and discussed followed by the results of the second group of tests.

For each group, the results of the first part of the recognizer, REC1, will be shown and briefly discussed. These will be followed by the results of REC2, which are far more interesting because they reflect the actual recognition time, and therefore will be an important factor in determining the usefulness of Early's algorithm. This section will include explanations of the time variances between input strings and between grammars. This will illustrate how the execution time of Early's algorithm is influenced not only by the length of the input string, but also by the characteristics of the input grammar.

Table 1 shows the times recorded when the Basis grammar subsets were used as input to REC1. For these particular grammars, the execution times increase linearly as more production rules are added to the grammars. The largest of the grammars, b.253, is the entire basis programming language grammar. Table 2 shows the times recorded when using the Eiffel grammar subsets as input to REC2. These, too, progress linearly. However, when Tables 1 and 2 are compared, it is evident that the variance is not strictly due to the number of productions. The reason for this is that REC1 must calculate the first sets of each left hand side after it reads in a grammar. The production rules in the grammar can make this either a simple or complicated procedure. The simplest case would be if all the production alternatives began with a terminal and the grammar contained no empty production rules. This is highly unlikely in a programming language grammar. When an alternative begins with a non-terminal, its first set must be found so it can be included in the first set of the left hand side. If the non-terminal derives an empty production, the next symbol in the right hand side must also be examined, which increases execution time. While these results are somewhat interesting, the work done in REC1 is preliminary and can be used repeatedly. Therefore it gives little or no information about the usefulness of Early's algortihm.

	REC1
b.05	0:00.7
b.10	0:01.1
b.15a	0:02.0
b.15b	0:01.5
b.20	0:02.3
b.22	0:02.7
b.27	0:03.1
b.34	0:03.7
b.41	0:04.0
b.48	0:04.7
b.55	0:05.1
b.aexp	0:01.2
b.lexp	0:02.5
b.253	1:12.0

TABLE 1

Execution times for REC1
using Basis grammars as input

	REC1
e.25	0:01.9
e.50	0:04.6
e.100	0:09.7
e.150	0:17.8
e.200	0:29.6
e.227	0:35.1

TABLE 2

Execution times for REC1
using Eiffel grammars as input

Tables 3-10 show the times recorded for REC2 for various input strings and programming language grammars. Each table shows how the times varied for a particular group of input strings. Recall that Earley claimed a complexity of O(n) for a given LR(k) grammar, where n was the length of the input string. Tables 3-6 illustrate this dependency very well. In general, they show that, for a particular LR grammar, the execution time doubles when the input string is doubled. Note that these tables show what happens when the input string is exactly doubled, not when only its length is doubled (see Appendix D). This result is nice, but somewhat expected, and there are more interesting points to discuss.

	b.a.3	b.a.7	b.a.14	b.a.28	b.a.56
b.05	0:01.9	0:02.6	0:04.2	0:07.3	0:13.5
b.10	0:04.1	0:06.4	0:10.9	0:19.9	0:37.9
b.15a	0:10.6	0:16.5	0:28.0	0:51.3	1:37.9
b.15b	0:03.9	0:08.4	0:15.0	0:28.2	0:54.6
b.20	0:10.5	0:18.8	0:32.7	1:00.7	1:57.0
b.22	0:11.5	0:20.5	0:35.8	1:06.6	2:08.7
b.27	0:13.9	0:23.3	0:41.0	1:16.6	2:28.7
b.34	0:13.5	0:34.8	1:04.1	2:03.3	4:02.9
b.41	0:13.4	1:16.0	2:26.7	4:48.6	9:39.3
b.48	0:20.4	0:45.6	1:24.7	2:37.8	5:09.5
b.55	0:20.6	1:27.5	2:46.7	5:26.9	10:52.7

TABLE 3

Execution times for REC2 using Basis grammars and input strings

	b.b.9	b.b.18	b.b.36	b.b.72
b.20	0:20.1	0:35.4	1:05.9	2:07.3
b.22	0:21.8	0:38.4	1:11.7	2:08.7
b.27	0:24.6	0:43.6	1:22.0	2:39.5
b.34	0:45.5	1:25.7	2:47.0	5:33.0
b.41	1:52.5	3:40.2	7:17.9	14:43.6
b.48	0:56.4	1:44.4	3:21.4	6:39.1
b.55	2:04.0	4:00.2	7:54.9	15:56.3

TABLE 4

Execution times for REC2 using Basis grammars and input strings

	b.c.11	b.c.22	b.c.44	b.c.88
b.27	0:25.4	0:35.3	1:25.3	2:45.7
b.34	0:46.3	1:27.5	2:50.6	5:39.5
b.41	1:53.8	3:42.7	7:23.7	14:52.7
b.48	0:57.2	1:46.1	3:25.2	6:46.1
b.55	2:15.2	4:02.2	7:58.5	16:02.9

TABLE 5

Execution times for REC2 using Basis grammars and input strings

	b.d.17	b.d.34	b.d.68	b.d.136
b.34	1:14.8	2:25.3	4:49.3	9:45.6
b.41	2:20.1	6:37.2	13:11.1	27:00.5
b.48	1:30.0	2:52.3	5:38.8	11:23.1
b.55	3:35.7	7:04.9	14:11.7	29:00.5

TABLE 6

Execution times for REC2 using Basis grammars and input strings

Notice in Table 3 how the times for grammars b.15a and b.15b differ. These two grammars contain the exact same number of production rules. The times shown are for the same input strings. Yet the execution times for grammar b.15a are almost twice those for b.15b. Here we see the dependence of the execution time for Earley's algorithm on the number of states created during the execution of the algorithm. The use of grammar b.15a results in the creation of 101, 165, 288 , 534 and 1026 states, respectively, for the 5 input strings shown in Table 3. Compare these with 51, 117, 212, 402, and 782 states for grammar b.15b. A graph of these results is shown in Figure 9. This same situation is apparent elsewhere in Tables 3-6. Compare the times recorded for grammars b.41 and b.48. These grammars were both derived from b.34. Seven rules were added to make b.41, and 14 different rules to make b.48. However, for these input strings, many more states were created during execution with b.41. For the input strings shown in Table 3, the numbers of states created using grammar b.48 were 154, 349, 642, 1228 and 2400, while the number of states created using b.41 were 126, 469, 894, 1744, and 3444. These results can be seen in Figure 10. It is evident here how great an effect the nature of the grammar can have on the achieved recognition time.

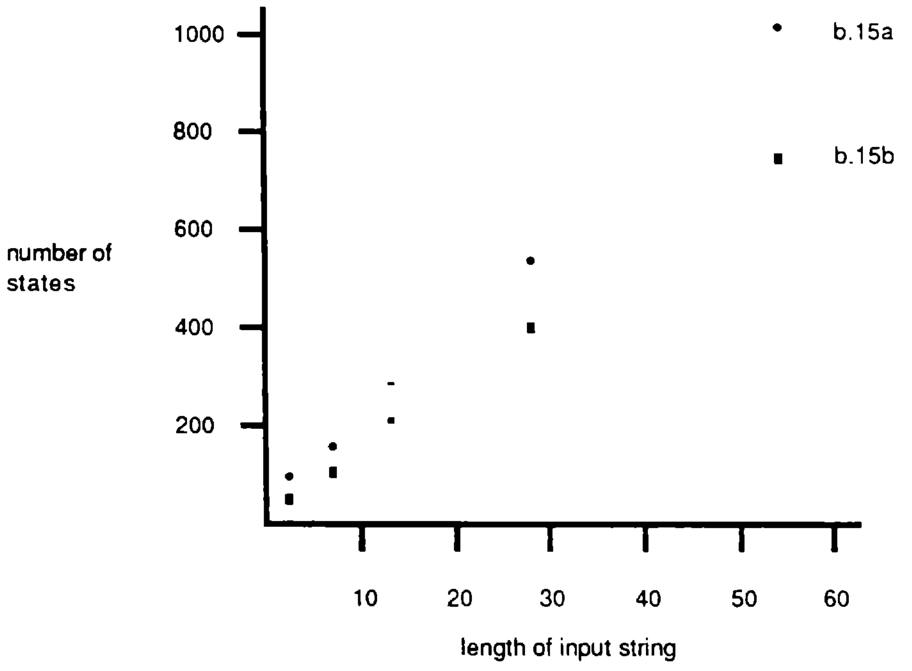


Figure 9

Graph of number of states generated by Earley's algorithm for grammars b.15a and b.15b and inputs b.a.3 through b.a.56

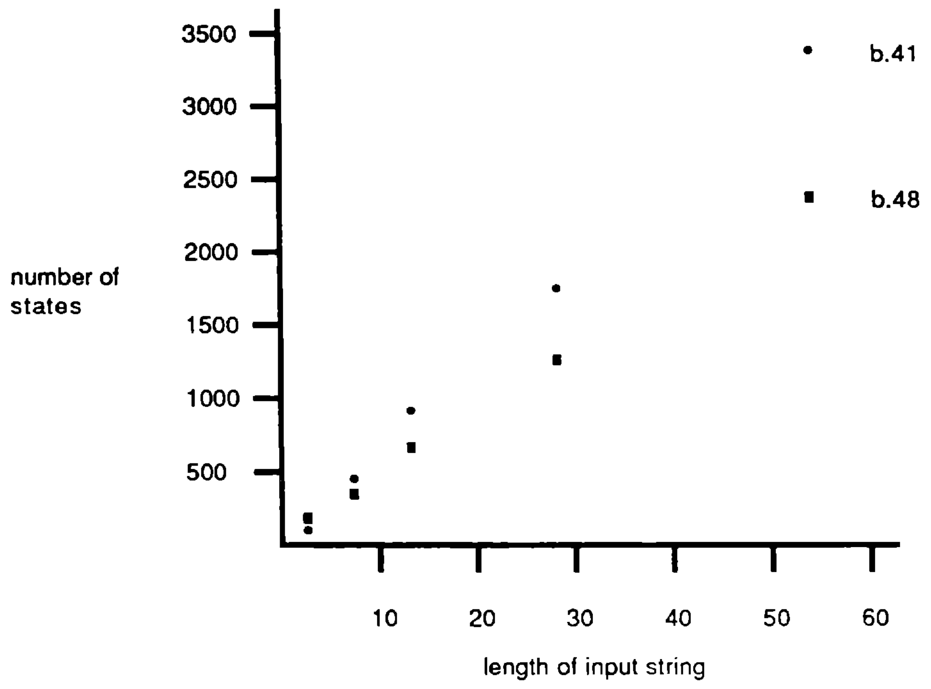


Figure 10

Graph of number of states generated by Earley's algorithm for grammars b.41 and b.48 and inputs b.a.3 through b.a.56

Recall that Earley calculated the complexity of his algorithm by determining the maximum number of states that could be created during its execution for a particular grammar. From the number of states created during the execution of Earley's algorithm for the grammars discussed above, one can determine a more precise complexity for Earley's algorithm. Given a grammar and an input string, the constants of the linear equation which describe the complexity of Earley's algorithm for an LR(1) grammar can be determined for this input. This is done by recording the number of states created during the execution of the recognizer for the input string, then doubling the input string and recording the number of states created during the execution of the recognizer for this doubled input string. A system of two linear equations is then solved. The solution reveals the values of the constants of the complexity of Early's algorithm for this grammar and input string. This complexity holds true for the given grammar and any input string that is a multiple of the input string used to determine the complexity. For the grammars b.15a, b.15b, b.41 and b.48, and input strings b.a.7, b.a.14, b.a.28 and b.a.56, the constants for the complexity of Earley's algorithm will now be determined.

For grammar b.15a and input strings b.a.7 through b.a.56, the number of states created produces the following equations:

$$165 = 7a + b \qquad (1)$$
$$288 = 14a + b \qquad (2)$$
$$534 = 28a + b \qquad (3)$$
$$1026 = 56a + b \qquad (4)$$

The values of a and b are determined by solving any two of the above equations simultaneously. Multiplying equation (1) by two and subtracting the equation (2) from the result produces:

$$42 = b$$

This gives the value for b. Substituting for b in equation (1) gives the value for a. The solution to the equations is

$$a = 17.57142857$$
$$b = 42$$

By substitution it can be shown that these values hold true for the remaining equations as well. Therefore, for this grammar and set of input strings, the complexity of Earley's algorithm is

$$17.57142857n + 42$$

where n is the length of the input string.

For grammar b.15b and input strings b.a.7 through b.a.56, the number of states created produces the following equations:

$117 = 7a + b \qquad (5)$

$212 = 14a + b \qquad (6)$

$402 = 28a + b \qquad (7)$

$782 = 56a + b \qquad (8)$

By solving any two of these equations simultaneously as was done above, it is determined that

$a = 13.57142857$

$b = 22$

Therefore the equation describing the complexity is

$$13.57142857n + 22.$$

For grammar b.41 and input strings b.a.7 through b.a.56, the number of states created produce the following equations:

$469 = 7a + b \qquad (13)$

$894 = 14a + b \qquad (14)$

$1744 = 28a + b \qquad (15)$

$3444 = 56a + b \qquad (16)$

Again by solving two of the above equations simultaneously, it is determined that

$a = 60.71428571$

$b = 44$

The equation describing the complexity is

$$60.71428571n + 44.$$

For grammar b.48 and input strings b.a.7 through b.a.56, the number of states created produces the following equations:

$349 = 7a + b$ (9)

$642 = 14a + b$ (10)

$1228 = 28a + b$ (11)

$2400 = 56a + b$ (12)

By solving two of the above equations simultaneously, it is determined that

$a = 41.85714286$

$b = 56$

The equation describing the complexity is

$41.85714286n + 56.$

In studying the grammars and input strings whose results are shown above, one notices that the portions of the grammars which describe arithmetic, logical, and relational expressions, and the input strings which contain these expressions, play a large part in increasing the execution time of the recognizer. Tables 7 and 8 show the results of using arithmetic and logical expression grammars along with short input strings as input to the recognizer. These tables illustrate the effect that expressions have on the recognizer. The high recognition times indicate that a large number of states are being created during the execution of the recognizer. This and the preceeding results illustrate the importance of the role which the grammar plays in the recognition process.

	b.e.1	b.e.3	b.e.7	b.b.15
b.aexp	0:11.6	0:20.1	0:37.0	1:11.1
b.lexp	1:30.9	2:13.2	3:38.0	6:28.9

TABLE 7
Execution times for REC2 using Basis expression grammars and input strings

	b.f.5	b.f.7	b.f.9	b.f.11
b.aexp	0:25.4	0:26.3	0:38.2	0:49.3
b.lexp	2:35.9	2:33.5	4:07.1	5:40.1

TABLE 8

Execution times for REC2 using Basis expression grammars and input strings

	e.a.3	e.a.7	e.a.15	e.a.31
e.25	0:02.1	0:05.5	0:11.4	0:23.2
e.50	0:02.5	0:07.0	0:14.6	0:30.2
e.100	0:05.4	0:11.7	0:23.5	0:47.0
e.150	0:05.5	0:11.9	0:23.8	0:47.7
e.200	0:05.9	0:12.3	0:24.2	0:47.9
e.227	0:07.4	0:16.8	0:36.6	1:14.2

TABLE 9

Execution times for REC2 using Eiffel grammars and input strings

	e.b.15	e.b.30	e.b.45	e.b.60
e.100	0:31.3	0:54.1	1:18.6	1:36.3
e.150	0:33.3	0:57.8	1:25.5	1:45.2
e.200	0:34.5	0:59.9	1:29.4	1:50.0
e.227	0:52.3	1:32.2	2:19.5	2:49.8

TABLE 10

Execution times for REC2 using Eiffel grammars and input strings

The results of using the Eiffel grammar subsets and input strings as inputs to the recognizer are shown in Tables 9 and 10. These grammars and input strings differ in two important ways from the basis grammars and input strings. First, the expression portions of the grammars were omitted until the grammar reached considerable size. This also means expressions were omitted from the input strings. Secondly, the Eiffel language is characterized by the use of keywords such as "class", "export", "inherit", "feature", etc. The result in Earley's algorithm is that fewer predictions are made until they are needed. Even the times for the larger grammars remain within a somewhat reasonable frame because the recognizer never gets to the point of having to predict many expressions that it will never need. This does not mean to imply that the Eiffel grammars will not make the recognizer grind when given an input string which contains an expression. It is only meant to illustrate that it is possible to find a reasonably sized grammar which provides acceptable results when used with Earley's algorithm.

	REC1
i.07	0:00.9
i.08	0:00.8
i.12	0:01.3
i.17	0:01.3
i.18	0:01.8
i.26	0:02.3
i.30	0:02.7
i.33	0:03.1

TABLE 11

Execution times for REC1 using command grammars as input

	REC1
c.06	0:00.5
c.15	0:01.1
c.21	0:01.5
c.29	0:02.3

TABLE 12

Execution times for REC1 using Config grammars as input

Now the results of the second group of tests using small, non-programming language grammars will be presented and briefly discussed. Tables 11, and 12 show the results of using the command and Config grammars as input to REC1. Like the previous REC1 results presented for the Basis and Eiffel grammar subsets, the execution times increase linearly as more production rules are added to the grammars. Once again, these results are not of particular interest, and are shown primarily for completeness.

Tables 13, 14, and 15 show the results of using the command grammars and input strings as input to REC2. Tables 16, 17 and 18 show the results of using the Config grammars and input strings as input to REC2. Recall that the input strings used in these particular tests do not double in content when they double in length as the previous input strings did. This is due to the nature of the programming tools and commands, and their corresponding grammars. Therefore, the different input strings contain different constructs. Earley's algorithm creates different numbers of states for these different constructs, and the differences are reflected in the recognition times. For this reason, the linearity of the results is not as apparent as it was when the programming language grammars and input strings were used as input to the recognizer. However, the poor performance of Earley's algorithm in these situations is quite apparent. Notice how quickly the recognition times increase when the size of the grammars increase in these examples. The tables show high recognition times for moderately sized grammars and input strings. From this information, it is doubtful that Earley's algorithm would be helpful in the implementation of programming tools. However, there is one more fact that must be considered in regards to the command grammars and input strings. This issue will be discussed in the following section.

	i.a.13	i.a.25
i.09	0:20.7	0:31.0
i.17	0:32.0	0:42.7
i.33	5:19.2	7:23.6

TABLE 13

Execution times for REC2 using command grammars and input strings

	i.b.16	i.b.28
i.17	0:31.3	0:41.5
i.33	4:59.5	7:01.7

TABLE 14

Execution times for REC2 using command grammars and input strings

	i.c.11	i.c.22	i.c.31	i.c.34	i.c.48
i.08	0:07.0	- - - - -	- - - - -	- - - - -	- - - - -
i.12	0:22.5	0:36.0	- - - - -	- - - - -	- - - - -
i.18	1:07.5	1:47.1	2:46.5	- - - - -	- - - - -
i.26	1:07.8	1:47.8	3:07.7	3:02.1	- - - - -
i.30	2:21.5	3:42.7	6:20.5	6:05.9	8:11.2
i.33	3:32.5	5:31.8	9:17.2	8:57.3	12:02.8

TABLE 15

Execution times for REC2 using command grammars and input strings

	c.a.4
c.06	0:00.7

TABLE 16

	c.b.5	c.b.11
c.15	0:14.2	0:57.6

TABLE 17

Execution times for REC2 using Config grammars and input strings

	c.c.7	c.c.13	c.c.19	c.c.25	c.c.32
c.21	0:45.8	1:26.5	1:31.8	- - - - -	- - - - -
c.29	2:37.2	3:17.6	4:18.9	5:21.2	6:31.4

TABLE 18

Execution times for REC2 using Config grammars and input strings

DISCUSSION

It is now time to evaluate the results presented in the previous section, and determine whether Earley's algorithm can be used successfully in any of the applications discussed earlier.

First, consider the use of Earley's algorithm in large, programming language applications such as a compiler-compiler or syntax-directed editor. From the results presented in the previous section, the conclusion must be drawn that these are not applications in which Earley's algorithm performs well. Recall that the purpose behind using Earley's algorithm in a compiler-compiler was twofold. First, it would allow LR(1) grammars to be expanded to include non-LR(1) constructs. Second, it would allow compiler writers the freedom to concentrate on implementation issues during compiler development. However, these results indicate that it would be more beneficial to bear the inconvenience of a LR(1) grammar and compiler-compiler than to use Earley's algorithm in a general context-free recognizer. This is primarily due to the lengths of the execution times of Earley's algorithm when given programming language grammars and input strings as input. The exception to this may be a syntax-directed editor. Since this is an interactive application, the poor performance of the machine will not be as evident when compared to the performance of the user. Consider that with some of the Eiffel grammars and input strings, the recognizer's speed is in the range of 25-75 symbols per minute. Compare this to the speed of an average typist, 20-60 words per minute. If symbols are equated with words, there is not much difference between the two. Perhaps this is a topic that should be explored in another investigation.

Now, consider the use of Earley's algorithm to parse input files to software tools, and to check command files for errors. It has been shown that for the programming tool used in this research, Earley's algorithm performed rather poorly. It is evident from these results that it would not be practical to use Earley's algortihm to assist in the implementation of this programming tool. The primary reason for wanting to use Earley's algorithm to assist in the implementation of programming tools was to simplify the job of the tools writer. However, it is better to inconvenience a programmer by forcing them to write additional code than to inconvenience an entire group of users with an inefficient and slow tool. It is not appropriate for users to suffer so that a tool writer's job can be simplified.

The recognizer also appears to have performed poorly when using command grammars as input. The purpose in using Earley's algorithm to parse command files is to catch errors before they cause problems. Since it is possible for a small list of commands to take hours to execute, a minute spent insuring that a command file is correct could save a great deal of time. However, the results indicate that the list of commands that could be checked using Earley's algorithm in a relatively short period of time is a small one, and therefore one that could probably be done easily by sight or other method.

From these results it can be seen that Earley's algorithm is useful in very few instances. Overall, the conclusion that must be drawn is that Earley's algorithm cannot be used successfully in a general context-free recognizer. The enormous number of states which must be created and processed during the execution of Earley's algorithm provide a tremendous amount of excess work, and therefore produce lengthy execution times for most realistic applications.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, U. S. A., 1974.

[2] A. V. Aho, R. Sethi and J. D. Ullman, Compilers: Principles,Techniques, and Tools, Addison-Wesley Publishing Company, U. S. A., 1985.

[3] A. V. Aho and J. D. Ullman, The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing, Prentice-Hall, U. S. A., 1972.

[4] R. C. Backhouse, Syntax of Programming Languages: Theory and Practice, Prentice-Hall, London, 1979.

[5] M. Bouckaert, A. Pirotte, and M. Snelling, "Efficient Parsing Algorithms for General Context-free Parsers," Information Sciences, Vol. 8, No. 1, Jan 1975, pp. 1-26.

[6] Y. T. Chiang and K. S. FU, "Parallel Parsing Algorithms and VLSI Implementations for Syntactic Pattern Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-6, No. 3, May 1984, pp. 302-314.

[7] T. W. Christopher, P. J. Hatcher and R.C. Kukuk, "Using Dynamic Programming to Generate Optimized Code in a Graham-Glanville Style Code Generator," Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Constuction, SIGPLAN Notices, Vol. 19, No. 6, June 1984, pp. 25-36.

[8] P. F. Dubois, and Z. C. Motteler, Basis User's Manual, M-189, Lawrence Livermore National Laboratory, U.S.A., 1987.

[9] J. Earley, An Efficient Context-Free Parsing Algorithm, thesis, Department of Computer Science, Carnegie-Mellon University, 1968.

[10] J. Earley, "An Efficient Context-Free Parsing Algorithm", Communications of the ACM, Vol. 13, No. 2, February, 1970, pp. 94-102.

[11] B. Meyer, Eiffel: A Language and Environment for Software Engineering, Interactive Software Engineering, Inc., U. S. A., 1987.

[12] B. Meyer, Eiffel Library Manual, Interactive Software Engineering, Inc., U. S. A., 1988.

[13] B. Meyer, Eiffel User's Manual, Interactive Software Engineering, Inc., U. S. A., 1988.

[14] B. Meyer, Object-oriented Software Construction, Prentice-Hall, Great Britain, 1988.

[15] K. O'Hair, The LR System User Manual, Lawrence Livermore National Laboratory, Livermore, CA, 1985.

[16] Programming Utilities for the Sun Workstation, Sun Microsystems, Inc., U. S. A., 1986.

[17] M. Snelling, "General Context-Free Parsing in Time N^2," International Computing Symposium 1973, A. Gunther et al.(eds), North-Holland Publishing Co., 1974, pp. 19-24.

[18] G. Sh. Vol'dman, "A Parsing Algorithm for Context-Sensitive Grammars," Programming and Computer Software, Vol. 7, No. 6, Nov-Dec 1981, pp. 302-307.

[19] G. A. Zaitman and O. A. Kholodenko, "Program For a Syntax Analyzer for Arbitrary CF-Grammars," Cybernetics, Vol. 14, No. 1, Jan-Feb 1978, pp. 36-40.

APPENDIX A -- Earley's Algorithm

Earley's algorithm is a recognition algorithm for general context-free grammars. For more detail see [9] and [10].

Earley's algorithm is based on the use of states and state sets.A state set is simply a group of states. A state consists of a production rule, a dot positioned on the right hand side of that production rule, a look-ahead string, and a pointer to a previous state set. To begin, the initial state is put into the first state set. The initial state has the form of:

```
X ::= . Z @        @        0
```

where Z is the goal symbol of the input grammar, @ is a unique look-ahead symbol, and 0 points to the first state set. The final state set will consist only of a final state. The final state has the form of:

```
X ::= Z @ .        @        0
```

If the final state set is reached, the string is considered valid. Otherwise it is not.

EARLEY'S ALGORITHM:

```
1)   Initialization
     Put the initial state into the first state set
2)   For each state set or until we reach the final state set do
         For each state in the set do
             If the dot is before a terminal, SCAN
             elsif the dot is before a non-terminal, PREDICT
             elsif the dot is at the end of the rule, COMPLETE
             endif
         endfor
     endfor
```

3) If the final state set has been reached, accept the string
else reject the string

SCAN

1) If the next input symbol matches the terminal following the dot, add a state to the next state set which is identical to the current state, except that the dot is advanced one position.

PREDICT

1) For each alternative of the non-terminal in the grammar, construct a new state with the dot at the beginning of the alternative's right hand side, and the pointer set to the current position in the linked list of state sets. The look ahead of the new state will be as follows:

   a) if a terminal follows the non-terminal in the current state, the look ahead will be that terminal

   b) if nothing follows the non-terminal in the current state, the look ahead will be the look ahead from the current state

   c) if a non-terminal follows the non-terminal in the current state, then a new state will be constructed for each terminal which could possibly occur next, with that terminal as the look ahead

COMPLETE

1) Compare the look ahead string of the current state with the next input symbol. If they match, look at the state set indicated by the pointer of that state. Find those states in that state set which have the left hand side of the current state after the dot. For each of these states, put a similar entry into the current state set, except advance the position of the dot by one.

APPENDIX B -- The Recognizer

Appendix B contains the Eiffel code of the recognizer. There are two parts to the recognizer, REC1 and REC2. In the first phase, execution begins with the Create feature of REC1. REC1 is responsible for reading the input grammar, processing it into its run-time structure, then storing the structure in a file. In the second phase, execution begins with the Create feature of REC2. REC2 retrieves the grammar structure, then processes the input string according to Earley's algorithm.

```
class REC1
-- This is a root class responsible for reading the input grammar into its run-time structure,
-- then storing that structure in the file grammar.store.
inherit
                STD_FILES
feature
                g:GRAMMAR;
Create is
do
        g.Create;                                       --create grammar object
        g.read_gr;                                      -- read grammar
        if (g.stop) then                                -- if an error occurred
                output.putstring("GRAMMAR PROCESSING FAILED");
                new_line;
        else    g.store("grammar.store");               -- store the grammar structure
                output.putstring("GRAMMAR PROCESSING COMPLETE");
                new_line;
        end;                    -- if-else
end;            -- feature Create
end;            -- class REC1

class GRAMMAR
-- This class represents a grammar and the operations which read the grammar from the input file
export
        r, stop, store, retrieve, read_gr
inherit
        STD_FILES ;
        STORABLE
feature
        r:S_LIST;                                       -- input grammar
        eog:BOOLEAN;                                    -- end of grammar
        eop:BOOLEAN;                                    -- end of production
        eoa:BOOLEAN;                                    -- end of alternative
        stop:BOOLEAN;                                   -- error status
        fin:BOOLEAN;
        infile:FILE;                                    -- grammar input file
        f_lst:SYMBOL_LIST;

get_sym:STRING  is
-- read the next symbol from the grammar input file
local
        temp:CHARACTER;
        t_str:STRING;                                   -- temporary variables
        done:BOOLEAN;
```

```
do
  from  done:=FALSE;
        t_str.Create(30);
  until  infile.end_of_file or done
  loop  from      temp:= infile.getchar;                     -- skip white space
        until     infile.end_of_file or temp /= '\n' and temp /= '\t' and temp /= '\r' and temp /= ' '
        loop      temp:=infile.getchar;                      -- temp <> whitespace
        end;      -- loop
        from                                                 -- get non-blank token
        until     infile.end_of_file or temp = '\n'  or temp = '\t' or temp = '\r' or temp = ' '
        loop      t_str.string_char(temp);
                  temp:=infile.getchar;
        end;
        if(t_str.same_as("&c")) then                         -- it's a comment
                  from
                  until      infile.end_of_file or temp = '\n' or temp = '\r'
                  loop       temp:=infile.getchar;
                  end;
                  t_str.clear;
        else      done:=TRUE;
        end;      -- if
  end;            -- loop
if (infile.end_of_file)  then
        stop:=TRUE;
end;
Result.Clone(t_str);
end;    -- feature get_sym

get_alt:SYMBOL_LIST      is
-- this feature collects the symbols of the next alternative of a production rule
local
    temp:CHARACTER;           t_sym:SYMBOL_LIST;
    t_str, ot_str:STRING;    i:INTEGER;
do
    from      t_sym.Create;
              t_str.Create(30);
              ot_str.Create(30);
              t_str:=get_sym;
              eoa:=FALSE;
    until     eoa or stop
    loop       if (t_str.entry(1) = '<' and then t_str.entry(t_str.length) = '>' and then t_str.length > 2) then
                     t_sym.insert_left(t_str);                       -- it's a non-terminal
                     t_str:=get_sym;
               else  if (t_str.same_as("&a") ) then
                         eoa:=TRUE;
                     elsif (t_str.same_as("&p")) then
                         eoa:=TRUE;
                         eop:=TRUE;
                     else  from    i:=1
                           until     i >= t_str.length
                           loop      if (t_str.entry(i) = '&') then
                                         if (t_str.entry(i+1) = '&' or  t_str.entry(i+1) = '<' or t_str.entry(i+1) = '>' ) then
                                             i:=i+1
                                         elsif (t_str.entry(i+1) = 'b') then
                                             i:=i + 1
                                             t_str.enter(i, ' ');
                                         else   putstring("ERROR:Term");
                                         end;
                                     end; -- if
                                     ot_str.string_char(t_str.entry(i));
                                      i:=i+1;
                           end; -- loop
```

```
                        if (i<=t_str.length) then
                             if (t_str.entry(i) = '&') then
                                 putstring("ERROR: last &");
                             else       ot_str.string_char(t_str.entry(i));
                             end;
                   end; -- if
                        t_sym.insert_left(ot_str.duplicate);
                        ot_str.clear;
                        t_str:=get_sym;
                   end; -- if-else
          end;      -- ifelse
   end;      -- loop
Result.Clone(t_sym);
end;      -- feature get_alt

get_rhs:LINKED_LIST[SYMBOL_LIST]          is
-- This feature collects all the possible alternatives (or right hand sides) which correspond
-- to the most recent left hand side.  If an error occurs, we must set stop:=TRUE;  We need to
-- look out for:
--              &c        comment                 &>           terminal >
--              &a        start alternative       &<           terminal <
--              &p        end production          &b           terminal blank
--              lambda productions                &&           terminal &
-- The alternatives will be placed in a linked list, which will be returned as the result.
local
       t_rhs:LINKED_LIST[SYMBOL_LIST];                                    -- temporary variables
       t_sym:SYMBOL_LIST;                        t_str:STRING;
do
       from     eop:=FALSE;
                t_rhs.Create;                    -- allocate temporary variables
                t_sym.Create;
       until    eop or stop
       loop     t_sym:=get_alt;
                t_sym.start;
                t_rhs.insert_left(t_sym);
       end;     -- loop
       Result.Clone(t_rhs);
       t_rhs.forget;
       t_sym.forget;
end;            -- feature get_rhs

get_lhs:STRING    is
-- This feature gets the left hand side of the next set of productions, if there is one.  It also looks for the end
--  of the grammar.  If an error occurs, we must set stop:=TRUE;  The lhs should have the form <non-terminal>.
-- Therefore as soon as we see <, we gobble up characters until we see >.
local
       t_str:STRING;                      -- temporary variables
       temp:CHARACTER;
       done:BOOLEAN;
do
       from     t_str.Create(30);                -- allocate t_str
                t_str:=get_sym;
       until    done or infile.end_of_file
       loop     if (t_str.same_as("&g")) then
                          eog:=TRUE;
                          done:=TRUE;
                elsif (t_str.entry(1) /= '<' or t_str.entry(t_str.length /='>') then
                          stop:=TRUE;
                          done:=TRUE;
                else      done:=TRUE;
                end;      -- if-elsif
       end;     -- loop
```

```
        if (stop) then
                output.putstring("ERROR: invalid grammar file."); new_line;
                output.putstring("Cannot find left hand side."); new_line;
        elsif (infile.eof) then
                stop:=TRUE;
                output.putstring("ERROR: unexpected end of file"); new_line;
        end;    -- if
        Result.Clone(t_str);
end;    -- feature get_lhs

first(x:STRING) is
local
        i,j,k,pos:INTEGER;          -- save position in grammar
        ts:STRING;
do
    pos:=r.position;
    r.go_find(x);
    if (r.offright) then
        output.putstring("ERROR: missing lhs.");
        output.putstring(x)
        new_line;
        stop:=TRUE;
    else  f_lst.insert_left(x);
        from    r.value.fst.start;

        until   r.value.fst.offright or stop
        loop    if (r.value.fst.value.entry(1) /= '<' or r.value.fst.value.entry(r.value.fst.value.length) /= '>'
                    of r.value.fst.value.length <= 2) then
                    first_term (pos);
                elsif (r.value.fst.value.same_as("<lambda>")) then
                    k:=r.value.fst.position;
                    r.mark;
                    r.go(pos);
                    from   r.value.rhs.start;
                    until  r.value.rhs.offright
                    loop   if (r.value.rhs.value.present(x)) then
                               from r.value.rhs.value.start
                               until r.value.rhs.value.offright
                               loop   if (r.value.rhs.value.value.same_as(x)) then
                                          first_lambda;
                                      end;
                                      r.value.rhs.value.forth;
                               end;     -- loop
                           end;     -- if-then
                           r.value.rhs.forth;
                    end;     -- loop
                    r.return;
                    r.value.fst.go(k);
                    r.value.fst.forth;
                elsif (not f_lst.present(r.value.fst.value)) then
                    i:=r.value.fst.position;
                    first(r.value.fst.value);
                    r.value.fst.go(i);
                    r.value.fst.delete;
                else r.value.fst.forth;
                end;     -- if-elsif
        end;     -- loop
        f_lst.del_left;
    end;     -- if-then-else
    r.go(pos);
 end;        -- feature FIRST
```

```
first_term (pos)  is
local
    ts:STRING;
    i, k:INTEGER;
do
    ts.Clone(r.value.fst.value)
    k:=r.value.fst.position;
    r.mark;
    r.go(pos);
    if (not r.value.fst.present(ts)) then
            fin:=FALSE;
            i:=r.value.fst.position;
            r.value.fst.finish;
            r.value.fst.insert_right(ts);
            r.value.fst.go(i);
    end;
    r.return;
    r.value.fst.go(k);
    r.value.fst.forth;
end;              -- feature first_term

first_lambda    is
local
    i, j:INTEGER;
do
    if (r.value.rhs.value.islast) then
            if (not r.value.fst.present("<lambda>")) then
                    i:=r.value.fst.position;
                    r.value.fst.finish;
                    r.value.fst.insert_right("<lambda>");
                    r.value.fst.go(i);
                    fin:=FALSE;
            end;        -- if
    else    i:=r.value.rhs.value.position + 1;
            if (not r.value.fst.present(r.value.rhs.value.i_th)) then
                    j:=r.value.fst.position;
                    r.value.fst.finish;
                    r.value.fst.insert_right(r.value.rhs.value.i_th(i));
                    r.value.fst.go(j);
                    fin :=FALSE;
            end;        -- if-then
    end;        -- if-else
end;              -- feature first_lambda
```

```
get_firsts  is
-- This feature finds all the firsts of each left hand side in the grammar.  It makes one pass
-- through the grammar collecting the terminals and non-terminals that appear as the first
-- symbols on all the right hand sides,  then uses the feature first to recursively break down
-- the non-terminals.
local
        cfst:STRING;
        j:INTEGER;
        t:SYMBOL_LIST;
do
   from   from      r.start;
                    t.Create;
          until     r.offright
          loop      t.wipe_out;
                    from      r.value.rhs.start;
                    until     r.value.rhs.offright
                    loop      if (r.value.rhs.value.empty) then
                              if (not t.present("<lambda>")) then
                                        t.insert_left("<lambda>");
                                    end;       -- if
                              else      if (not t.present(r.value.rhs.value.first)) then
                                            t.insert_left(r.value.rhs.value.first.duplicate);
                                        end;
                              end;
                              r.value.rhs.forth;
                    end;      -- loop
   until  fin
   loop   fin:=TRUE;
          from      r.finish;
          until     r.offleft or stop
          loop      from r.value.fst.start;
                    until r.value.fst.offright or stop
                    loop cfst.Clone(r.value.fst.value);
                         if (cfst.entry(1) = '>' and cfst.entry(cfst.length) = '>'
                                    and not cfst.same_as("<lambda>") and cfst.length > 2) then
                              j:=r.value.fst.position;
                              first(cfst);
                              r.value.fst.go(j);
                              r.value.fst.delete;
                         else  r.value.fst.forth;
                         end;       -- if
                    end;      -- loop
                    r.back;
          end;      -- loop
   end;       -- loop
end;            -- feature GET_FIRSTS

feature gr_init   is
-- This feature reads the grammar from the input file
local
    t_rule, t_two:RULE;               -- temporary variables
    t_lhs:STRING;
    t_s:SYMBOL_LIST;
    t_rhs:LINKED_LIST[SYMBOL_LIST];
do
    from   r.Create;                               -- allocate grammar structure
           infile.open_read;                       -- open grammar input file
           f_lst.Create;
           t_two.Create;                           -- allocate temporary variables
           t_rule.Create;
           t_lhs.Create(30);
           t_rhs.Create;
           t_lhs:= get_lhs                     -- read in a  lhs
```

```
                if (not stop and not eog) then
                      t_rhs:=get_rhs;                        -- read in alternatives of lhs
                      if (not stop) then
                              t_rule.assign(t_lhs, t_rhs);
                              t_two.Clone(t_rule);
                              r.insert(t_two);               -- put in grammar
                      end;          -- if
                end;          -- if
                if (not stop) then
                      t_rhs.wipe_out;
                      t_s.Create;
                      t_s.insert_right("@@@");
                      t_s.insert_right(r.first.lhs);
                      t_rhs.insert_right(t_s);
                      t_rule.assign("<%%%>", t_rhs);
                      t_two.Clone(t_rule);
                      r.insert(t_two);
                end;
        until   eog or stop
        loop    t_lhs:=get_lhs;              -- read in lhs of grammar
                if (not stop and not eog) then
                      t_rhs:=get_rhs;
                      if (not stop) then
                              t_rule.assign(t_lhs, t_rhs);
                              t_two.Clone(t_rule);
                              r.insert(t_two);
                      end;          -- if
                end;          -- if
        end;          -- loop
   end;               -- feature gr_init

   read_gr            is
   -- This feature will attempt to open the grammar input file and read the input grammar into the
   -- feature r.  If an error occurs which prohibits it from successfully reading the grammar, the feature
   -- stop will be set to TRUE.  Stop is checked by REC1 to see if the grammar was successfully read.
   do
        infile.Create("z.bnf");     -- allocate grammar input file
        if (not infile.exists) then
                output.putstring("ERROR: Grammar file does not exist.");
                new_line;
                stop:=TRUE;
        elsif (not infile.readable) then
                output.putstring("ERROR: Grammar file is not readable.");
                new_line;
                stop:=TRUE;
        else    output.putstringl("READING GRAMMAR");
                new_line;
                gr_init;
                output.putstring("PROCESSING GRAMMAR");
                new_line;
                if (not stop) then
                      get_firsts;
                end;          -- if
                infile.close;
        end;
   end;               -- feature READ_GR
   end;               class GRAMMAR
```

```
class S_LIST
export
            back,                go_find,         nb_elements,
            delete,              insert,          offleft,
            duplicate,           isfirst,         offright,
            empty,               islast,          position,
            finish,              i_th,            return,
            first,               last,            start,
            forth,               mark,            value,
            go,                  merge,           wipe_out
inherit
            LINKED_LIST[RULE]
                 redefine present;
feature

present(x:RULE):BOOLEAN                 is
local
    done:BOOLEAN;
    hi,lo,look:INTEGER;
do
from        hi:=nb_elements;
            lo:=1;
until       done or hi < lo
loop        look:= (hi + lo) div 2;
            if (i_th(look).eq(x)) then
                 done:=TRUE;
            else  if (i_th(look).lt(x)) then
                       lo:=look + 1;
                 else       hi:= look-1;
                 end;       -- if-else
            end;       -- if-else
end;             -- loop
if (done) then
    Result:=TRUE;
else        Result:=FALSE;
end;             -- if-else
end;             -- feature  PRESENT

go_find(x:STRING)          is
local
    done:BOOLEAN;                      hi,lo,look:INTEGER;
do
from        hi:=nb_elements;
            lo:=1;
until       done or hi < lo
loop        look:= (hi + lo) div 2;
            if (i_th(look).lhs.same_as(x)) then
                 done:=TRUE;
                 go(look);
            else  if (i_th(look).lhs.le(x)) then
                       lo := look + 1;
                 else   hi:= look - 1
                 end;       -- if-else
            end;            -- if-else
end;             -- loop
if (not done) then
    go(nb_elements);
    forth;
end;
end;             -- feature GO_FIND
```

```
insert(x:RULE)  is
local
    done:BOOLEAN;                                hi,lo,look,z:INTEGER;
do
    z:=position;
    if(empty) then
            insert_right(x);
    elsif (first.gt(x)) then
            go(1);
            insert_left(x);
            done:=TRUE;
    elsif(last.lt(x)) then
            go(nb_elements);
            insert_right(x);
            done:=TRUE;
    else          from  hi:=nb_elements;
                        lo:=1;
                  until  i_th(lo).le(x) and i_th(hi).ge(x) and hi <= lo + 1
                  loop  look:= (hi + lo) div 2
                        if (i_th(look_.lt(x)) then
                              lo:=look;
                        else  hi:=look;
                        end;          -- if-else
                  end;          -- loop
                  go(lo);
                  insert_right(x);
    end;          -- if-elsif-elsif-else
    go(z);
end;                        -- feature insert
end;              -- class S_LIST

class RULE
-- This class represents a set of production rules which correspond to a single left hand side.
-- lhs is the non-terminal on the left hand side
-- fst is a linked list of the possible "firsts" of the lhs
-- rhs is a linked list of all the possible right hand alternatives
export
    lhs,                      -- lhs of grammar production
    fst,                      -- firsts of ohs
    rhs,                      -- alternatives of lhs
    assign,                   -- assign values to lhs and rhs
    set_first      ,          -- initialize fst
    eq,le,lt,ge,gt            -- compare left hand sides
inherit
    COMPARABLE  redefine le,gt,ge
feature
    lhs:STRING;
    fst:SYMBOL_LIST;
    rhs:LINKED_LIST[SYMBOL_LIST];

assign(left:STRING; right:LINKED_LIST[SYMBOL_LIST])  is
-- assign left to lhs and right to rhs
do
            lhs.Clone(left);
            rhs.Clone(right);
end;              -- feature assign
```

```
set_first (f:SYMBOL_LIST) is
-- initialize fst to f
do
            fst.clone(f);
end;            -- feature set_first

eq(x:RULE):BOOLEAN is
-- tell if the current rule is the same as x
do
    if (lhs.same_as(x.lhs)) then
            Result:=TRUE;
    else    Result:=FALSE;
    end;
end;            -- feature eq
gt(x:RULE):BOOLEAN      is
-- tell if current rule is greater than x
do
    if (lhs.gt(x.lhs)) then
            Result:=TRUE;
    else    Result:=FALSE;
    end;
end;            -- feature GT

le(x:RULE) : BOOLEAN is
-- tell if current rule is less than or equal to x
do
    if (lhs.le(x.lhs)) then
            Result:= TRUE;
    else    Result:=FALSE;
    end;
end;

ge(x:RULE):BOOLEAN is
-- tell if current rule is greater than or equal to x
do
    if (lhs.same_as(x.lhs) or lhs.gt(x.lhs)) then
            Result:=TRUE;
    else    Result:=FALSE;
    end;
end;

lt(x:RULE): BOOLEAN     is
-- tell if current rule is less than x
do
    if (lhs.same_as(x.lhs) or lhs.gt(s.lhs)) then
            Result:=FALSE;
    else    Result:=TRUE;
    end;
end;

Create is
do
    lhs.Create(30);
    rhs.Create;
    fst.Create;
end;        -- feature Create
end;            -- class RULE

class SYMBOL_LIST
-- A SYMBOL_LIST is a linked list of strings used to represent the right hand side of a
-- production rule in the input grammar.  It is also used to represent the first sets.  For each
-- possible non-terminal, its first set is represented by a SYMBOL_LIST
```

```
export
          back,                 forth,              nb_elements,
          change_i_th,          go,                 offleft,
          change_value,         insert_left,        offright,
          delete,               insert_right,       position,
          del_left,             isfirst,            present,
          delete_right,         islast,             return,
          duplicate,            i_th,               start,
          empty,                last,               value,
          finish,               merge_left,         wipe_out,
          first,                merge_right
inherit
                LINKED_LIST[STRING]
                        redefine present
feature

del_left is
do
        position:=position - 1;
        nb_elements:=nb_elements - 1;
end;

present (x:STRING):BOOLEAN                            is
-- tell whether the string x is present in the symbol_list
local
        z:INTEGER;
do
        from    z:= position;
                start;
        until   offright or x.same_as(value)
        loop    forth;
        end;    -- from-loop
        if (not offright) then
                Result:= TRUE;
        else    Result:= FALSE;
        end;    -- if-else
        go(z);
end;            -- feature present

end;            -- class SYMBOL_LIST

class REC2
inherit STD_FILES
feature
        g:GRAMMAR;                              -- input grammar
        infile:FILE;
        als:LINKED_LIST[STATE_SET];             -- list of all state sets
        next:STRING;                            -- next input symbol
        cs:STATE;                               -- current state
        css:STATE_SET;                          -- current state set
        nss:STATE_SET;                          -- next state set
        s1:STATE;                               -- first state
        ss1:STATE_SET;                          -- first state set
        lss:STATE_SET;                          -- lambda-rule state set
        rss:STATE_SET;                          -- recursive state set
        c_rhs:SYMBOL_LIST;                      -- current right hand side
        t:STRING is "@@@";-- teminator symbol
        halt:BOOLEAN;                           -- indicates error

Initialize is
local
        g_file:FILE;
        add:BOOLEAN;
```

```
do
        output.putstring("Initializing...");                    new_line;
        g_file.Create("grammar.store");
        if(g_file.exists and g_file.readable) then
                  g.Create;
                  g:=g.retrieve("grammar.store");
                  infile.Create("z.input");
                  if (infile.exists and infile.readable) then
                            infile.open_read;
                            als.Create;                                   -- allocate all states
                            ss1.Create;                                   -- create first state set
                            add:=ss1.add_state(1,1,1,t,1);                -- create initial state
                            als.insert_right(ss1);                        -- insert first state set
                            nss.Create;                                   -- allocate next state set
                            lss.Create;
                            rss.Create;
                            next.Create(10);
                  else      output.putstring("FATAL ERROR: cannot read input file");
                            new_line;
                            halt:=TRUE;
                  end;      -- if
end;    -- initialize

Create is
local
        k:INTEGER;
        tss:STATE_SET;
do
initialize;
if(not halt) then
        output.putstring("Processing...");    new_line;
        from      als.start;
        until     als.offright or else als.value.isfinal or else halt
        loop      next:=get_next;
                  css.Clone(als.value);
                  nss.wipe_out;
                  lss.wipe_out;
                  rss.wipe_out;
                  from      css.start;
                  until     css.offright or halt
                  loop      cs.Clone(css.value);
                            c_rhs.Clone(g.r.i_th(cs.lp).rhs.i_th(cs.rp));
                            operate;
                            css.forth;
                  end;      -- loop
                  if (not halt) then
                            css.start;
                            nss.start;
                            als.change_value(css.duplicate(css.nb_elements));
                            if (nss.nb_elements /= 0) then
                                      als.insert_right(nss.duplicate(ncc.nb_elements));
                            end;    -- if
                  end;      -- if
                  als.forth;
        end;      -- loop
        if (als.offright or halt) then
                  output.putstring(" ERROR: invalid input string");
                  new_line;
        else      output.putstring("DONE: string is valid.");
                  new_line;
        end;      -- if
 end    -- if
 end;   -- feature Create
```

```
get_next:STRING  is
local
        temp:CHARACTER;  -- temporary variables
        t_str:STRING;
        done:BOOLEAN;
do
        t_str.Create(30);
        from    temp:= infile.getchar              -- skip whit space
        until   infile.end_of_file or temp /= '\n' and temp /= '\t' and temp /= '\r' and temp /= ' '
        loop    temp:=infile.getchar;              -- temp <> whitespace
        end;    -- loop
        from              -- get non-blank token
        until   infile.end_of_file or temp  = '\n' or temp = '\t' or temp = '\r' or temp = ' '
        loop    t_str.string_char(temp);
                temp:=infile.getchar;
        end;    -- loop
        if (infile.end_of_file and t_str.length = 0) then
                Result:= ("@@@");
        else    Result.Clone(t_str);
        end;
end;    -- get_next

operate is
-- test to see if we need to scan, predict or complete and call appropriate feature
do
if (cs.dot > c_rhs.nb_elements or else c_rhs.nb_elements = 0) then
        complete;  -- if the dot's at the end
elsif (c_rhs.i_th(cs.dot).entry(1) /= '<' or else c_rhs.i_th(cs.dot).entry(c_rhs.i_th(cs.dot).length) /= '>'
        or else    c_rhs.i_th(cs.dot).length < 3) then    -- . non-terminal
                   scan;
else    if (not rss.there(cs)) then
                   predict;
        end;
end;    -- if
end;    -- feature operate

scan is
-- apply scanner to current state
local
        tstate:STATE;
        ts:STATE;
        i:BOOLEAN;
do
if (c_rhs.i_th(cs.dot).same_as(next)) then                -- if (terminal = next)
        i:= nss.add_state(cs.lp,cs.rp,cs.dot+1,cs.look,cs.point);
end;
end;    -- feature scan

predict is
-- apply predictor to current state
local
        tstr:STRING;                                  tstate, ts:STATE;
        t_alt:LINKED_LIST[SYMBOL_LIST];               t_list:SYMBOL_LIST;
        i, j:INTEGER;                                 add, done:BOOLEAN;
do
tstr.Clone(c_rhs.i_th(cs.dot));                       -- the non-terminal to the right of dot
if (tstr.same_as(g.r.i_th(cs.lp).lhs)) then
        add:=rss.add_state(cs.lp,cs.rp,cs.dot,cs.look,cs.point);
end;    -- if
g.r.go_find(tstr);
if (g.r.offright) then
        output.putstring("ERROR: no alternatives found for non-terminal");           new_line;
        halt:=TRUE;
```

```
else    t_alt.Clone(g.r.value.rhs);                          -- a linked list of rh sides
        from    t_alt.start;
        until   t_alt.offright
        loop    from    j:=1;
                        done:=FALSE;
                until   done
                loop    done:=TRUE;
                        if (cs.dot+j > c_rhs.nb_elements) then      -- non-term is last
                              add:=css.add_state(g.r.position,t_alt.position,1,cs.look,als.position);
                        elsif (c_rhs.i_th(cs.dot+j).entry(1) /= '<' or
                        c_rhs.i_th(cs.dot_j).entry(c_rhs.i_th(cs.dot_j).length) /= '>' or
                              c_rhs.i_th(cs.dot_j).length < 3) then     -- it's a terminal
                              add:= css.add_state(g.r.position, t_alt.position,1, c_rhs.i_th(cs.dot+j),als.position);
                        else  tstr.Clone(c_rhs.i_th(cs.dot+j));                  -- non-term . non-term , use firsts
                              g.r.mark;
                              g.r.go_find(tstr);
                              t_list.Clone(g.r.value.lst);
                              g.r.return;
                              from    t_list.start;
                              until   t_list.offright
                              loop    if (not t_list.value.same_as("<lambda>") ) then
                                            add:=css.add_state(g.r.position, t_alt.position,1, t_list.value, als.position);
                                            if (add and not lss.empty and not t_alt.value.empty) then
                                                          check_lambda;
                                            end;
                                      else              done:=FALSE;
                                                        j:=j + 1;
                                      end;              -- if-else
                                      t_list.forth;
                              end;  -- from
                        end; -- if-elsif-else
                end;    -- loop
                t_alt.forth;
        end;    -- loop
end;    -- if
end;            -- feature predict

complete is
-- apply completer to current state
local
        d, l, r, i, j:INTEGER;                           ts2, ts:STATE;
        str:STRING;                                      add:BOOLEAN;
do
if (cs.look.same_as(next)) then
        if (not f.r.i_th(cs.lp).rhs.i_th(cs.rp).empty)  then
                from    als.mark;
                        als.go(cs.point);
                        als.value.start;
                until   als.value.offright
                loop    d:=als.value.value.dot;
                        l:=als.value.value.lp;
                        r:=als.value.value.rp;
                        if (d <= g.r.i_th(l).rhs.i_th(r).nb_elements) then
                              if (g.r.i_th(cs.lp).lhs.same_as(g.r.i_th(l).rhs.i_th(r).i_th(d))) then
                                    add:=css.add_state(l, r, d+1, als.value.value.look, als.value.value.point);
                                    if (d+1<= g.r.i_th(l).rhs.i_th(r).nb_elements) then
                                          str.Clone(g.r.i_th(l).rhs.i_th(r).i_th(d+1));
                                          if (str.entry(1) = '<' and str.entry(str.length) = '>' and
                                                      str.length > 2 and not str.same_as("lambda>") then
                                                            check_lambda;
                                          end;
                                    end;
```

```
                                end;   -- if
                            end; -- if
                            als.value.forth;
                        end;    -- loop
                        als.return;
                else    from    add:=lss.add_state(cs.lp, cs.rp, cs.dot, cs.look, cs.point);
                                j:=css.position;
                                css.start;
                        until   css.offright
                        loop    d:=css.value.dot;
                                l:=css.value.lp;
                                r:=css.value.rp;
                                if (d <= g.r.i_th(l_.rhs.i_th(r).nb_elements) then
                                    if (g.r.i_th(cs.lp).lhs.same_as(g.r.i_th(l).rhs.i_th(r).i_th(d))) then
                                        add:=css.add_state(l, r, d+1, css.value.look, css.value.point);
                                        if (d+1 <= g.r.i_th(l).rhs.i_th(r).nb_elements) then
                                            str.Clone(f.r.i_th(l).rhs.i_th(r).i_th(d+1));
                                            if (str.entry(1) = '<' and str.entry(str.length) = '>' and
                                                    str.length > 2 and not str.same_as("<lambda>")) then
                                                        check_lambda;
                                            end;        -- if
                                        end;            -- if
                                    end;   -- if
                                end; -- if
                                css.forth;
                        end;    -- loop
                        css.go(j);
                end;    -- if
end;    -- if
end;    -- feature complete

check_lambda  is
-- check to see if the state being added has one of the lambda rules to the right of the dot
-- if it does, add a new state in the same manner that the completer does
-- note that this must be recursive, because there can be nested lambdas
local   str:STRING;
        x, z:INTEGER;
        add:BOOLEAN;
do
from    z:=css.position;
        css.finish;
        x:= 0;
until   x = css.nb_elements
loop    x := css.nb_elements
        from    lss.start;
        until   lss.offright                                    -- "complete" and check_lambda"
        loop    if (g.r.i_th(lss.value.lp).lhs.same_as(g.r.i_th(css.value.lp).rhs.i_th(css.value.rp.i_th(css.value.dot))) then
                        add:=css.add_state(css.value.lp, css.value.rp, css.value.dot+1, css.value.look, css.value.point);
                        if (css.value.dot+1 <= g.r.i_th(css.value.lp).rhs.i_th(css.value.rp).nb_elements) then
                            str.Clone(g.r.i_th(css.value.lp).rhs.i_th(css.value.rp).i_th(css.value.dot+1));
                            if (str.entry(1) = '<' and str.entry(str.length) = '>' and str.length > 2 and
                                    not str.same_as("<lambda>")) then
                                    check_lambda;
                            end;  -- if
                        end; -- if
                end;    -- if
                lss.forth;
        end;    -- loop
end;    -- loop
css.go(z);
end;    -- feature check_lambda
end; -- class REC2
```

```
class STATE_SET
-- A STATE_SET represents a state set in Earley's Algorithm.
-- It is a linked list of STATEs to be processed in order.
export
                add_state,          isfinal,            offright,
                back,               isfirst,            present,
                duplicate,          islast,             position,
                empty,              i_th,               return,
                finish,             last,               start,
                first,              mark,               value,
                forth,              nb_elements,        wipe_out,
                go,                 offleft,            there
inherit
                LINKED_LIST[STATE]
                            redefine present
feature

there(s:STATE) :BOOLEAN         is
local
        z:INTEGER;              done:BOOLEAN;
do
from    z:=position;
        start;
unti    offright or else done
loop    if (value.lp = s.lp and then value.rp = s.rp and then value.dot = s.dot and then value.point = s.point) then
                done:=TRUE;
        else    forth;
        end;    -- if-else
end;    --loop
if (done) then
        Result:=TRUE;
else    Result:=FALSE;
end;    -- if-else
go(z);
end;                -- feature there


add_state(i,j,k:INTEGER,l:STRING,p:INTEGER) :BOOLEAN      is
local  ts,tstate:STATE;
        z:INTEGER;
do
        tstate.Create(i,j,k,l,p);
        if (not present (tstate)) then
                z:= position;
                finish;
                ts.Clone(tstate);
                insert_right(ts);
                go(z);
                Result:=TRUE;
        else    Result:=FALSE;
        end;
end;
```

```
present (s:STATE):BOOLEAN       is
-- tells whether s is present in the state set
local
        z:INTEGER;
        done:BOOLEAN;
do
from    z:=position;
        start;
until   offright or else done
loop    if (value.lp = s.lp and then value.rp = s.rp and then value.dot = s.dot and then
                    value.look.same_as(s.look) and then value.point = s.point) then
                    done:=TRUE;
        else        forth;
        end;        -- if-else
end;    -- loop
if (done) then
        Result:=TRUE;
else    Result:=FALSE;
end;    -- if-else
go(z);
end;                -- feature present

isfinal:BOOLEAN  is
-- tells if the state set is final
do
        if(nb_elements = 1 and then first.lp = 1 and then first.rp = 1 and then first.dot = 3 and then
                    first.lokk.same_as("@@@") and then first.point = 1) then
                    Result:=TRUE;
        else        Result:=FALSE;
        end;        -- if-else
end;                -- feature isfinal
end;    -- class STATE_SET

class STATE
-- A STATE represents a state in Earley's Algorithm. A state in Earley's Algorithm consists of a
-- production rule with a dot positioned in the right hand side (rhs) of the rule, a look-ahead
-- string, and a pointer to a state set
--      lp is an integer pointer to the left hand side (lhs) of a production
--                  rule in the input grammar
--      rp is an integer pointer to a rhs associated with the above lhs
--      dot indicates the position of the dot in the rhs
--      look is the look-ahead string
--      point is the pointer to a state set
export
        lp,                 -- lhs of state
        rp,                 -- rhs of state
        dot,                -- position of dot in state
        look,               -- look-ahead string
        point               -- pointer to state set
feature
        lp, rp, dot, point:INTEGER;
        look:STRING;

Create(i:INTEGER, j:INTEGER, k:INTEGER, s:STRING, l:INTEGER)      is
do
        lp:=i;
        rp:=j;
        dot:=k;
        look.Create(30);
        point:=l;
end;                -- feature Create
end;                -- class STATE
```

APPENDIX C -- Input Grammars

The following grammars are those which describe subsets of the Basis programming language.

b.05

```
<parse>        ::=   <stlist>
<stlist>       ::=   <stlist> <stmt> ;
               ::=
<stmt>         ::=   name = name
               ::=   integer name
```

b.10

```
<parse>        ::=   <stlist>
<stlist>       ::=   <stlist> <stmt> <eos>
               ::=
<eos>          ::=   ;
<stmt>         ::=   <id> = <primitive>
               ::=   <type> name
               ::=
<type>         ::=   integer
<primitive>    ::=   <id>
<id>           ::=   name
```

b.15a

```
<parse>        ::=   <stlist>
<stlist>       ::=   <stlist> <stmt> <eos>
               ::=
<eos>          ::=   ;
<stmt>         ::=   <id> = <primitive>
               ::=   <type> name
               ::=
<type>         ::=   integer
               ::=   real
               ::=   logical
               ::=   complex
               ::=   chameleon
               ::=   character
<primitive>    ::=   <id>
<id>           ::=   name
```

b.15b

```
<parse>        ::=   <stlist>
<stlist>       ::=   <stlist> <stmt> <eos>
               ::=
<eos>          ::=   ;
<stmt>         ::=   <id> = <exp>
               ::=   <type> name
               ::=
<type>         ::=   integer
```

```
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=   - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <id>
<id>            ::=   name
```

b.20

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
<stmt>          ::=   <id> = <exp>
                ::=   <type> name
                ::=
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=   - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <id>
<id>            ::=   name
```

b.22

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
<stmt>          ::=   <assign>
                ::=   <misc>
                ::=
<assign>        ::=   <id> = <exp>
<misc>          ::=   <type> name
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=   - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <id>
<id>            ::=   name
```

b.27

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
```

```
<stmt>          ::=   <assign>
                ::=   <misc>
                ::=
<assign>        ::=   <id> = <exp>
<misc>          ::=   <type> <varlist>
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
<varlist>       ::=   <dname> <vals>
                ::=   <varlist> , <dname> <vals>
<dname>         ::=   name
<vals>          ::=   = <exp>
                ::=
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=   - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <id>
<id>            ::=   name
```

<u>b.34</u>

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
<stmt>          ::=   <assign>
                ::=   <misc>
                ::=
<assign>        ::=   <id> = <exp>
<misc>          ::=   <type> <varlist>
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
<varlist>       ::=   <dname> <vals>
                ::=   <varlist> , <dname> <vals>
<dname>         ::=   name
<vals>          ::=   = <exp>
                ::=
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=   - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <term> * <primry>
                ::=   <term> / <primry>
                ::=   <primry>
<primry>        ::=   <factor> ** <factor>
                ::=   <factor>
<factor>        ::=   <primitive>
                ::=   ( <exp> )
<primitive>     ::=   <id>
<id>            ::=   name
```

b.41

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
<stmt>          ::=   <assign>
                ::=   <misc>
                ::=
<assign>        ::=   <id> = <lexp>
<misc>          ::=   <type> <varlist>
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
<varlist>       ::=   <dname> <vals>
                ::=   <varlist> , <dname> <vals>
<dname>         ::=   name
<vals>          ::=   = <lexp>
                ::=
<lexp>          ::=   <exp> < <exp>
                ::=   <exp> <= <exp>
                ::=   <exp> > <exp>
                ::=   <exp> >= <exp>
                ::=   <exp> = <exp>
                ::=   <exp> <> <exp>
                ::=   <exp>
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=    - <term>
                ::=   + <term>
                ::=   <term>
<term>          ::=   <term> * <primry>
                ::=   <term> / <primry>
                ::=   <primry>
<primry>        ::=   <factor> ** <factor>
                ::=   <factor>
<factor>        ::=   <primitive>
                ::=   ( <lexp> )
<primitive>     ::=   <id>
<id>            ::=   name
```

b.48

```
<parse>         ::=   <stlist>
<stlist>        ::=   <stlist> <stmt> <eos>
                ::=
<eos>           ::=   ;
<stmt>          ::=   <assign>
                ::=   <misc>
                ::=   <plot>
                ::=
<assign>        ::=   <id> = <exp>
<misc>          ::=   <type> <varlist>
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character
```

```
<varlist>         ::=   <dname> <vals>
                  ::=   <varlist> , <dname> <vals>
<dname>           ::=   name
<vals>            ::=   = <exp>
                  ::=
<plot>            ::=   plot <scale> <exp> , <exp> <label>
                  ::=   plot <scale> <exp> , <exp> , <exp> <label>
                  ::=   plot <scale> <exp> , <exp> , <exp> , <exp>
                  ::=   plot <scale> <exp> <label>
                  ::=   plotm <scale> <exp> , <exp> , <exp>
<scale>           ::=   linlin
                  ::=   linlog
                  ::=   loglin
                  ::=   loglog
                  ::=   equal
                  ::=
<label>           ::=   @ <exp>
                  ::=
<exp>             ::=   <exp> + <term>
                  ::=   <exp> - <term>
                  ::=   - <term>
                  ::=   + <term>
                  ::=   <term>
<term>            ::=   <term> * <primry>
                  ::=   <term> / <primry>
                  ::=    <primry>
<primry>          ::=   <factor> ** <factor>
                  ::=   <factor>
<factor>          ::=   <primitive>
                  ::=   ( <exp> )
<primitive>       ::=   <id>
<id>              ::=   name
```

<u>b.55</u>

```
<parse>           ::=   <stlist>
<stlist>          ::=   <stlist> <stmt> <eos>
                  ::=
<eos>             ::=   ;
<stmt>            ::=   <assign>
                  ::=   <misc>
                  ::=   <plot>
                  ::=
<assign>          ::=   <id> = <lexp>
<misc>            ::=   <type> <varlist>
<type>            ::=   integer
                  ::=   real
                  ::=   logical
                  ::=   complex
                  ::=   chameleon
                  ::=   character
<varlist>         ::=   <dname> <vals>
                  ::=   <varlist> , <dname> <vals>
<dname>           ::=   name
<vals>            ::=   = <lexp>
                  ::=
<plot>            ::=   plot <scale> <exp> , <exp> <label>
                  ::=   plot <scale> <exp> , <exp> , <exp> <label>
                  ::=   plot <scale> <exp> , <exp> , <exp> , <exp>
                  ::=   plot <scale> <exp> <label>
                  ::=   plotm <scale> <exp> , <exp> , <exp>
```

```
<scale>          ::=   linlin
                 ::=   linlog
                 ::=   loglin
                 ::=   loglog
                 ::=   equal
                 ::=
<label>          ::=   @ <exp>
                 ::=
<lexp>           ::=   <exp> < <exp>
                 ::=   <exp> <= <exp>
                 ::=   <exp> > <exp>
                 ::=   <exp> >= <exp>
                 ::=   <exp> = <exp>
                 ::=   <exp> <> <exp>
                 ::=   <exp>
<exp>            ::=   <exp> + <term>
                 ::=   <exp> - <term>
                 ::=   - <term>
                 ::=   + <term>
                 ::=   <term>
<term>           ::=   <term> * <primry>
                 ::=   <term> / <primry>
                 ::=   <primry>
<primry>         ::=   <factor> ** <factor>
                 ::=   <factor>
<factor>         ::=   <primitive>
                 ::=   ( <lexp> )
<primitive>      ::=   <id>
<id>             ::=   name
```

b.253

```
<parse>          ::=   <stlist>
<stlist>         ::=   <stlist> <stmt> <eos>
                 ::=
<nonnullstlist>  ::=   <nonnullstmt> <eos> <stlist>
<eos>            ::=   semicolon
                 ::=   cr
<simplestmt>     ::=   <assign>
                 ::=   <list>
                 ::=   <display>
                 ::=   <misc>
                 ::=   <plot>
<structstmt>     ::=   <dostmt>
                 ::=   <forstmt>
                 ::=   <whilestmt>
                 ::=   <ifstmt>
<nonnullstmt>    ::=   <simplestmt>
                 ::=   <structstmt>
                 ::=   <funcspec>
                 ::=
<stmt>           ::=   <nonnullstmt>
                 ::=   { }
                 ::=   whitespace
                 ::=   ^
                 ::=
<funcspec>       ::=   functions <funcdes> <eos> <stlist> endf
<funcdes>        ::=   name
                 ::=   reference ( <paramlist> )
<paramlist>      ::=   name
                 ::=   <paramlist> , name
<dostmt>         ::=   do <id> = <dorange> <eos> <stlist> enddo
                 ::=   do <eos> <stlist> <docontrol>
```

```
<dorange>       ::=   <first> , <last> <incr>
<first>         ::=   <exp>
<last>          ::=   <exp>
<incr>          ::=   , <exp>
<docontrol>     ::=   enddo
                ::=   until ( <lexp> )
<forstmt>       ::=   for ( <forinit> , <forout> , <lstmt> ) <stlist> endfor
<forinit>       ::=   <assignlist>
                ::=
<forout>        ::=   <lexp>
<assignlist>    ::=   <assign>
                ::=   <assignlist> <eos> <assign>
<lstmt>         ::=   <cstmt>
<cstmt>         ::=   <stmt>
                ::=   <cstmt> <eos> <stmt>
<whilestmt>     ::=   while <whilexp> <stlist> endwhile
<whilexp>       ::=   ( <lexp> )
<ifstmt>        ::=   if <ifexp> <lalt>
                ::=   if <ifexp> <thenlist> <optelse> endif
<ifexp>         ::=   ( <lexp> )
<lalt>          ::=   <eos> <nonnullstmt>
                ::=   <nonnullstmt>
<thenlist>      ::=   <eos> then <stlist>
                ::=   then <stlist>
<optelse>       ::=   else <stlist>
                ::=   elsif <ifexp> <alt2> <optelse>
                ::=
<alt2>          ::=   <nonnullstlist>
                ::=   <eos> <nonnullstlist>
                ::=   <eos> then <stlist>
                ::=   then <stlist>
<misc>          ::=   remark <exp>
                ::=   box name
                ::=   tv <exp>
                ::=   tv
                ::=   tek <exp>
                ::=   tek
                ::=   package name
                ::=   run <runspec>
                ::=   generate <genspec>
                ::=   step <runspec>
                ::=   finish <finspec>
                ::=   read <filename>
                ::=   forget <forgetlist>
                ::=   forget
                ::=   push <pkg>
                ::=   pop
                ::=   timer <exp>
                ::=   output to <filename>
                ::=   save to <filename>
                ::=   save <savelist>
                ::=   call <id>
                ::=   call <rid> <args>
                ::=   <id> command <comlist>
                ::=   <scope> <type> <varlist>
                ::=   break <optlevel>
                ::=   next <optlevel>
                ::=   <indevice> <inlist>
                ::=   <outdevice> <outlist>
                ::=   return <optval>
<scope>         ::=   global
                ::=
<indevice>      ::=
                ::=   <exp>
<outdevice>     ::=   <exp>
                ::=
                ::=   plot
<outlist>       ::=   << <lexp>
                ::=   << return
                ::=   <outlist> << <lexp>
                ::=   <outlist> << return
<inlist>        ::=   >> <lhs>
                ::=   >> return
                ::=   <inlist> >> <lhs>
                ::=   <inlist> >> return
<assign>        ::=   <lhs> = <lexp>
<lhs>           ::=   <id>
                ::=   <rid> <args>
<list>          ::=   list
                ::=   list <listspec>
<plot>          ::=   plot <scale> <exp> , <exp> <label>
                ::=   plot <scale> <exp> , <exp> , <exp> <label>
                ::=   plot <scale> <exp> , <exp> , <exp> , <exp>
                ::=   plot <scale> <exp> <label>
                ::=   plotm <scale> <exp> , <exp> , <exp>
<optlevel>      ::=   <level>
                ::=
<level>         ::=   integer-constant
                ::=   hex-constant
                ::=   octal-constant
                ::=   ( integer-constant )
                ::=   ( hex-constant )
                ::=   ( octal-constant )
<runspec>       ::=   <pkg> <flag> <count>
<genspec>       ::=   <pkg> <flag>
<finspec>       ::=   <pkg> <flag>
<flag>          ::=
                ::=   plotonly
                ::=   plot
                ::=   noplot
<count>         ::=
                ::=   integer-constant
                ::=   hex-constant
                ::=   octal-constant
<varlist>       ::=   <dvar> <vals>
                ::=   <varlist> , <dvar> <vals>
<display>       ::=   <ditem>
                ::=   <display , <ditem>
<savelist>      ::=
                ::=   <sitem>
                ::=   <savelist> , <sitem>
<forgetlist>    ::=   <dname>
                ::=   <forgetlist> , <dname>
<dvar>          ::=   <dname>
                ::=   <dname> <args>
<dname>         ::=   name
                ::=   reference
                ::=   ' <exp> '
<vals>          ::=   = <lexp>
                ::=
<ditem>         ::=   <exp>
                ::=   <group>
<listspec>      ::=   <listelt>
                ::=   <listspec> , <listelt>
                ::=   <listspec> <listelt>
```

```
<listelt>       ::=   variables
                ::=   groups
                ::=   packages
                ::=   functions
                ::=   <id>
                ::=   <group>
<optval>        ::=   <lexp>
                ::=
<type>          ::=   integer
                ::=   real
                ::=   logical
                ::=   complex
                ::=   chameleon
                ::=   character * <exp>
                ::=   character
                ::=   indirect
<sitem>         ::=   <id>
                ::=   <group>
<scale>         ::=   linlin
                ::=   linlog
                ::=   loglin
                ::=   loglog
                ::=   equal
                ::=
<label>         ::=   @ <exp>
                ::=
<explist>       ::=   <lexp>
                ::=   <explist> , <lexp>
<lexp>          ::=   <lexp> | <lterm>
                ::=   <lterm>
<lterm>         ::=   <lterm> & <lprimary>
                ::=   <lprimary>
<lprimary>      ::=   ~ <lfactor>
                ::=   <lfactor>
<lfactor>       ::=   <exp> < <exp>
                ::=   <exp> <= <exp>
                ::=   <exp> > <exp>
                ::=   <exp> >= <exp>
                ::=   <exp> = <exp>
                ::=   <exp> == <exp>
                ::=   <exp> <> <exp>
                ::=   <exp> ~= <exp>
                ::=   <exp> ? [ <exp> , <exp> ]
                ::=   <exp> ? [ <exp> , <exp> )
                ::=   <exp> ? ( <exp> , <exp> ]
                ::=   <exp> ? ( <exp> , <exp> )
                ::=   <exp>
<exp>           ::=   <exp> + <term>
                ::=   <exp> - <term>
                ::=    - <term>
                ::=    + <term>
                ::=   <term>
<term>          ::=   <term> * <primry>
                ::=   <term> * ! <primry>
                ::=   <term> / <primry>
                ::=   <term> ! <primry>
                ::=   <term> // <primry>
                ::=   <term> / ! <primry>
                ::=   <primry>
<primry>        ::=   <factor> ** <factor>
                ::=   <factor>
```

```
<factor>         ::=    <primitive>
                 ::=    <factor> <args>
                 ::=    <factor> '
                 ::=    [ <explist> ]
                 ::=    ( <lexp> )
<args>           ::=    ( <arglist> )
<arglist>        ::=    <argitem>
                 ::=    <argitem> , <arglist>
<comlist>        ::=    <comitem>
                 ::=    <comitem> , <comlist>
                 ::=    <comitem> whitespace <comlist>
<comitem>        ::=    <filename>
<filename>       ::=    fcomstr
                 ::=    <argitem>
<argitem>        ::=    <dexp>
                 ::=    & <id>
                 ::=    & <rid> <args>
                 ::=    <dexp> : <dexp>
<dexp>           ::=
                 ::=    <lexp>
<primitive>      ::=    <id>
                 ::=    <rid> <args>
                 ::=    string
                 ::=    real-constant
                 ::=    complex-constant
                 ::=    integer-constant
                 ::=    hex-constant
                 ::=    octal-constant
<id>             ::=    name
                 ::=    name . name
<rid>            ::=    reference
                 ::=    name . reference
<group>          ::=    Groupname
                 ::=    name . Groupname
<pkg>            ::=    name
```

b.lexp

```
<parse>          ::=    <lexp>
<lexp>           ::=    <lexp> | <lterm>
                 ::=    <lterm>
<lterm>          ::=    <lterm> & <lprimary>
                 ::=    <lprimary>
<lprimary>       ::=    ~ <lfactor>
                 ::=    <lfactor>
<lfactor>        ::=    <exp> < <exp>
                 ::=    <exp> <= <exp>
                 ::=    <exp> > <exp>
                 ::=    <exp> >= <exp>
                 ::=    <exp> = <exp>
                 ::=    <exp> <> <exp>
                 ::=    <exp>
<exp>            ::=    <exp> + <term>
                 ::=    <exp> - <term>
                 ::=    - <term>
                 ::=    + <term>
                 ::=    <term>
<term>           ::=    <term> * <primry>
                 ::=    <term> / <primry>
                 ::=    <primry>
<primry>         ::=    <factor> ** <factor>
                 ::=    <factor>
```

```
<factor>            ::=   <primitive>
                    ::=   ( <lexp> )
<primitive>         ::=   name
```

<u>b.aexp</u>

```
<parse>             ::=   <exp>
<exp>               ::=   <exp> + <term>
                    ::=   <exp> - <term>
                    ::=    - <term>
                    ::=   + <term>
                    ::=    <term>
<term>              ::=   <term> * <primry>
                    ::=   <term> / <primry>
                    ::=   <primry>
<primry>            ::=   <factor> ** <factor>
                    ::=   <factor>
<factor>            ::=   <primitive>
                    ::=   ( <exp> )
<primitive>         ::=   name
```

The following grammars are those which describe subsets of the Eiffel programming language. The complete Eiffel grammar, e.227, was converted to this notation from the grammar shown in [13].

<u>e.025</u>

```
<class_declaration>              ::=   <class_header> <formal_generics> <exports> <parents>
                                       <features> <class_invariant> end
<class_header>                   ::=   class <class_name>
<class_name>                     ::=   identifier
<formal_generics>                ::=
<feature_name>                   ::=   identifier
<exports>                        ::=
<parents>                        ::=
<type>                           ::=   INTEGER
                                 ::=   REAL
<features>                       ::=
                                 ::=   feature <feature_declaration_list>
<feature_declaration_list>       ::=
                                 ::=   <feat_decl>
<feat_decl>                      ::=   <feature_declaration>
                                 ::=   <feat_decl> ; <feature_declaration>
<feature_declaration>            ::=   <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>               ::=
<type_mark>                      ::=   : <type>
                                 ::=
<feature_value_mark>             ::=
                                 ::=   is <feature_value>
<feature_value>                  ::=   <constant>
<constant>                       ::=   integer
                                 ::=   real
<class_invariant>   ::=
```

e.50

```
<class_declaration>            ::=   <class_header> <formal_generics> <exports> <parents>
                                     <features> <class_invariant> end
<class_header>                 ::=   <deferred_mark> class <class_name>
<deferred_mark>                ::=
                               ::=   deferred
<class_name>                   ::=   identifier
<formal_generics>              ::=
<feature_name>                 ::=   identifier
<exports>                      ::=
<parents>                      ::=
<type>                         ::=   INTEGER
                               ::=   BOOLEAN
                               ::=   CHARACTER
                               ::=   REAL
<features>                     ::=
                               ::=   feature <feature_declaration_list>
<feature_declaration_list>     ::=
                               ::=   <feat_decl>
<feat_decl>                    ::=   <feature_declaration>
                               ::=   <feat_decl> ; <feature_declaration>
<feature_declaration>          ::=   <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>             ::=
                               ::=   ( <entity_declaration_list> )
<entity_declaration_list>      ::=
                               ::=   <ent_decl_list>
<ent_decl_list>                ::=   <entity_declaration_group>
                               ::=   <ent_decl_list> ; <entity_declaration_group>
<entity_declaration_group>     ::=   <id_list> : <type>
<id_list>                      ::=   identifier
                               ::=   <id_list> , identifier
<type_mark>                    ::=   : <type>
                               ::=
<feature_value_mark>           ::=
                               ::=   is <feature_value>
<feature_value>                ::=   <constant>
                               ::=   <routine>
<constant>                     ::=   <integer_constant>
                               ::=   <character_constant>
                               ::=   <boolean_constant>
                               ::=   <real_constant>
                               ::=   <string_constant>
<integer_consant>              ::=   <sign> integer
<sign>                         ::=
                               ::=   -
<character_constant>           ::=   ' character '
<boolean_constant>             ::=   true
                               ::=   false
<real_constant>                ::=   <sign> real
<string_constant>              ::=   " string "
<routine>                      ::=
<class_invariant>              ::=
```

e.100

```
<class_declaration>            ::=   <class_header> <formal_generics> <exports> <parents>
                                     <features> <class_invariant> end
<class_header>                 ::=   <deferred_mark> class <class_name>
<deferred_mark>                ::=
                               ::=   deferred
<class_name>                   ::=   identifier
```

```
<formal_generics>          ::=
                           ::=   [ <formal_generic_list> ]
<formal_generic_list>      ::=
                           ::=   <f_gen_list>
<f_gen_list>               ::=   <formal_generic>
                           ::=   <f_gen_list> , <formal_generic>
<formal_generic>           ::=   <formal_generic_name> <constraint>
<formal_generic_name>      ::=   identifier
<constraint>               ::=
                           ::=   -> <class_type>
<exports>                  ::=
                           ::=   export <export_list>
<export_list>              ::=
                           ::=   <ex_list>
<ex_list>                  ::=   <export_item>
                           ::=   <ex_list> , <export_item>
<export_item>              ::=   <feature_name> <export_restriction>
<feature_name>             ::=   identifier
<export_restriction>       ::=
                           ::=   { <class_list> }
<class_list>               ::=
                           ::=   <c_list>
<c_list>                   ::=   <class_name>
                           ::=   <c_list> , <class_name>
<parents>                  ::=
                           ::=   inherit <parent_list>
<parent_list>              ::=
                           ::=   <p_list>
<p_list>                   ::=   <parent>
                           ::=   <p_list> ; <parent>
<parent>                   ::=   <class_type> <rename_clause> <redefine_clause>
<class_type>               ::=   <class_name> <actual_generics>
<actual_generics>          ::=
                           ::=   [ <type_list> ]
<type_list>                ::=
                           ::=   <t_list>
<t_list>                   ::=   <type>
                           ::=   <t_list> , <type>
<type>                     ::=   INTEGER
                           ::=   BOOLEAN
                           ::=   CHARACTER
                           ::=   REAL
                           ::=   <class_type>
                           ::=   <formal_generic_name>
                           ::=   <assosciation>
<association>              ::=   like <anchor>
<anchor>                   ::=   <feature_name>
                           ::=   Current
<rename_clause>            ::=
                           ::=   <rename <rename_list>
<rename_list>              ::=
                           ::=   <ren_list>
<ren_list>                 ::=   <rename_pair>
                           ::=   <ren_list> , <rename_pair>
<rename_pair>              ::=   <feature_name> as <feature_name>
<redefine_clause>          ::=
                           ::=   redefine <feature_list>
<feature_list>             ::=
                           ::=   <feat_list>
<feat_list>                ::=   <feature_name>
                           ::=   <feat_list> , <feature_name>
<features>                 ::=
                           ::=   feature <feature_declaration_list>
```

```
<feature_declaration_list>      ::=
                                ::=   <feat_decl>
<feat_decl>                     ::=   <feature_declaration>
                                ::=   <feat_decl> ; <feature_declaration>
<feature_declaration>           ::=   <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>              ::=
                                ::=   ( <entity_declaration_list> )
<entity_declaration_list>       ::=
                                ::=   <ent_decl_list>
<ent_decl_list>                 ::=   <entity_declaration_group>
                                ::=   <ent_decl_list> ; <entity_declaration_group>
<entity_declaration_group>      ::=   <id_list> : <type>
<id_list>                       ::=   identifier
                                ::=   <id_list> , identifier
<type_mark>                     ::=   : <type>
                                ::=
<feature_value_mark>            ::=
                                ::=   is <feature_value>
<feature_value>                 ::=   <constant>
<constant>                      ::=   <integer_constant>
                                ::=   <character_constant>
                                ::=   <boolean_constant>
                                ::=   <real_constant>
                                ::=   <string_constant>
<integer_consant>               ::=   <sign> integer
<sign>                          ::=
                                ::=   -
<character_constant>            ::=   ' character '
<boolean_constant>              ::=   true
                                ::=   false
<real_constant>                 ::=   <sign> real
<string_constant>               ::=   " string "
<class_invariant>               ::=
```

<u>e.150</u>

```
<class_declaration>             ::=     <class_header> <formal_generics> <exports> <parents>
                                        <features> <class_invariant> end
<class_header>                  ::=     <deferred_mark> class <class_name>
<deferred_mark>                 ::=
                                ::=     deferred
<class_name>                    ::=     identifier
<formal_generics>               ::=
                                ::=     [ <formal_generic_list> ]
<formal_generic_list>           ::=
                                ::=     <f_gen_list>
<f_gen_list>                    ::=     <formal_generic>
                                ::=     <f_gen_list> , <formal_generic>
<formal_generic>                ::=     <formal_generic_name> <constraint>
<formal_generic_name>           ::=     identifier
<constraint>                    ::=
                                ::=     -> <class_type>
<exports>                       ::=
                                ::=     export <export_list>
<export_list>                   ::=
                                ::=     <ex_list>
<ex_list>                       ::=     <export_item>
                                ::=     <ex_list> , <export_item>
<export_item>                   ::=     <feature_name> <export_restriction>
<feature_name>                  ::=     identifier
<export_restriction>            ::=
                                ::=     { <class_list> }
```

```
<class_list>                   ::=
                               ::=   <c_list>
<c_list>                       ::=   <class_name>
                               ::=   <c_list> , <class_name>
<parents>                      ::=
                               ::=   inherit <parent_list>
<parent_list>                  ::=
                               ::=   <p_list>
<p_list>                       ::=   <parent>
                               ::=   <p_list> ; <parent>
<parent>                       ::=   <class_type> <rename_clause> <redefine_clause>
<class_type>                   ::=   <class_name> <actual_generics>
<actual_generics>              ::=
                               ::=   [ <type_list> ]
<type_list>                    ::=
                               ::=   <t_list>
<t_list>                       ::=   <type>
                               ::=   <t_list> , <type>
<type>                         ::=   INTEGER
                               ::=   BOOLEAN
                               ::=   CHARACTER
                               ::=   REAL
                               ::=   <class_type>
                               ::=   <formal_generic_name>
                               ::=   <assosciation>
<association>                  ::=   like <anchor>
<anchor>                       ::=   <feature_name>
                               ::=   Current
<rename_clause>                ::=
                               ::=   <rename <rename_list>
<rename_list>                  ::=
                               ::=   <ren_list>
<ren_list>                     ::=   <rename_pair>
                               ::=   <ren_list> , <rename_pair>
<rename_pair>                  ::=   <feature_name> as <feature_name>
<redefine_clause>              ::=
                               ::=   redefine <feature_list>
<feature_list>                 ::=
                               ::=   <feat_list>
<feat_list>                    ::=   <feature_name>
                               ::=   <feat_list> , <feature_name>
<features>                     ::=
                               ::=   feature <feature_declaration_list>
<feature_declaration_list>     ::=
                               ::=   <feat_decl>
<feat_decl>                    ::=   <feature_declaration>
                               ::=   <feat_decl> ; <feature_declaration>
<feature_declaration>          ::=   <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>             ::=
                               ::=   ( <entity_declaration_list> )
<entity_declaration_list>      ::=
                               ::=   <ent_decl_list>
<ent_decl_list>                ::=   <entity_declaration_group>
                               ::=   <ent_decl_list> ; <entity_declaration_group>
<entity_declaration_group>     ::=   <id_list> : <type>
<id_list>                      ::=   identifier
                               ::=   <id_list> , identifier
<type_mark>                    ::=   : <type>
                               ::=
<feature_value_mark>           ::=
                               ::=   is <feature_value>
<feature_value>                ::=   <constant>
                               ::=   <routine>
```

```
constant>                          ::=   <integer_constant>
                                   ::=   <character_constant>
                                   ::=   <boolean_constant>
                                   ::=   <real_constant>
                                   ::=   <string_constant>
<integer_consant>                  ::=   <sign> integer
<sign>                             ::=
                                   ::=   +
                                   ::=   -
<character_constant>               ::=   ' character '
<boolean_constant>                 ::=   true
                                   ::=   false
<real_constant>                    ::=   <sign> real
<string_constant>                  ::=   " string "
<routine>                          ::=   <precondition> <externals> <local_variables> <body>
                                         <postcondtion> <rescue> end
<precondition>                     ::=
<assertion>                        ::=
                                   ::=   <assert_list>
<assert_list>                      ::=   <assertion_clause>
                                   ::=   <assert_list> ; <assertion_clause>
<assertion_clause>                 ::=   <tag_mark> <unlabeled_assertion_clause>
<tag_mark>                         ::=
                                   ::=   <tag> :
<tag>                              ::=   identifier
<unlabeled_assertion_clause>       ::=   <boolean_expression>
                                   ::=   <comment>
<boolean_expression>               ::=   <expression>
<comment>                          ::=   -- string
<externals>                        ::=
                                   ::=   <ext_decl>
<ext_decl>                         ::=   <external_declaration>
                                   ::=   <ext_decl> ; <external_declaration>
<external_declaration>             ::=   <feature_name> <formal_arguments> <type_mark>
                                         <external_name> <language>
<language>                         ::=   language <string_constant>
<external_name>                    ::=
                                   ::=   name <string_constant>
<local_variables>                  ::=
                                   ::=   local <entity_declaration_list>
<body>                             ::=   <full_body>
<full_body>                        ::=   <normal_body>
<normal_body>                      ::=   do <compound>
<compound>                         ::=
                                   ::=   <comp_list>
<comp_list>                        ::=   <instruction>
                                   ::=   <comp_list> ; <instruction>
<instruction>                      ::=   <assignment>
                                   ::=   <check>
                                   ::=   <retry>
                                   ::=   <debug>
<assignment>                       ::=   <entity> := <expression>
<entity>                           ::=   identifier
                                   ::=   Result
<expression>                       ::=
                                   ::=   <constant>
                                   ::=   <entity>
<debug>                            ::=   debug <compound> end
<check>                            ::=   check <assertion> end
<retry>                            ::=   retry
<postcondition>                    ::=
<rescue>                           ::=
<class_invariant>                  ::=
```

<u>e.200</u>

```
<class_declaration>       ::=   <class_header> <formal_generics> <exports> <parents>
                                <features> <class_invariant> end
<class_header>            ::=   <deferred_mark> class <class_name>
<deferred_mark>           ::=
                          ::=   deferred
<class_name>              ::=   identifier
<formal_generics>         ::=
                          ::=   [ <formal_generic_list> ]
<formal_generic_list>     ::=
                          ::=   <f_gen_list>
<f_gen_list>              ::=   <formal_generic>
                          ::=   <f_gen_list> , <formal_generic>
<formal_generic>          ::=   <formal_generic_name> <constraint>
<formal_generic_name>     ::=   identifier
<constraint>              ::=
                          ::=   -> <class_type>
<exports>                 ::=
                          ::=   export <export_list>
<export_list>             ::=
                          ::=   <ex_list>
<ex_list>                 ::=   <export_item>
                          ::=   <ex_list> , <export_item>
<export_item>             ::=   <feature_name> <export_restriction>
<feature_name>            ::=   identifier
<export_restriction>      ::=
                          ::=   { <class_list> }
<class_list>              ::=
                          ::=   <c_list>
<c_list>                  ::=   <class_name>
                          ::=   <c_list> , <class_name>
<parents>                 ::=
                          ::=   inherit <parent_list>
<parent_list>             ::=
                          ::=   <p_list>
<p_list>                  ::=   <parent>
                          ::=   <p_list> ; <parent>
<parent>                  ::=   <class_type> <rename_clause> <redefine_clause>
<class_type>              ::=   <class_name> <actual_generics>
<actual_generics>         ::=
                          ::=   [ <type_list> ]
<type_list>               ::=
                          ::=   <t_list>
<t_list>                  ::=   <type>
                          ::=   <t_list> , <type>
<type>                    ::=   INTEGER
                          ::=   BOOLEAN
                          ::=   CHARACTER
                          ::=   REAL
                          ::=   <class_type>
                          ::=   <formal_generic_name>
                          ::=   <assosciation>
<association>             ::=   like <anchor>
<anchor>                  ::=   <feature_name>
                          ::=   Current
<rename_clause>           ::=
                          ::=   rename <rename_list>
<rename_list>             ::=
                          ::=   <ren_list>
<ren_list>                ::=   <rename_pair>
                          ::=   <ren_list> , <rename_pair>
<rename_pair>             ::=   <feature_name> as <feature_name>
```

```
<redefine_clause>                ::=
                                 ::=    redefine <feature_list>
<feature_list>                   ::=
                                 ::=    <feat_list>
<feat_list>                      ::=    <feature_name>
                                 ::=    <feat_list> , <feature_name>
<features>                       ::=
                                 ::=    feature <feature_declaration_list>
<feature_declaration_list>       ::=
                                 ::=    <feat_decl>
<feat_decl>                      ::=    <feature_declaration>
                                 ::=    <feat_decl> ; <feature_declaration>
<feature_declaration>            ::=    <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>               ::=
                                 ::=    ( <entity_declaration_list> )
<entity_declaration_list>        ::=
                                 ::=    <ent_decl_list>
<ent_decl_list>                  ::=    <entity_declaration_group>
                                 ::=    <ent_decl_list> ; <entity_declaration_group>
<entity_declaration_group>       ::=    <id_list> : <type>
<id_list>                        ::=    identifier
                                 ::=    <id_list> , identifier
<type_mark>                      ::=    : <type>
                                 ::=
<feature_value_mark>             ::=
                                 ::=    is <feature_value>
<feature_value>                  ::=    <constant>
                                 ::=    <routine>
<constant>                       ::=    <integer_constant>
                                 ::=    <character_constant>
                                 ::=    <boolean_constant>
                                 ::=    <real_constant>
                                 ::=    <string_constant>
<integer_consant>  :             :=     <sign> integer
<sign>                           ::=
                                 ::=    +
                                 ::=    -
<character_constant>             ::=    ' character '
<boolean_constant>               ::=    true
                                 ::=    false
<real_constant>                  ::=    <sign> real
<string_constant>                ::=    " string "
<routine>                        ::=    <precondition> <externals> <local_variables> <body>
                                        <postcondtion> <rescue> end
<precondition>                   ::=
                                 ::=    require <assertion>
<assertion>                      ::=
                                 ::=    <assert_list>
<assert_list>                    ::=    <assertion_clause>
                                 ::=    <assert_list> ; <assertion_clause>
<assertion_clause>               ::=    <tag_mark> <unlabeled_assertion_clause>
<tag_mark>                       ::=
                                 ::=    <tag> :
<tag>                            ::=    identifier
<unlabeled_assertion_clause>     ::=    <boolean_expression>
                                 ::=    <comment>
<boolean_expression>             ::=    <expression>
<comment>                        ::=    -- string
<externals>                      ::=
                                 ::=    <ext_decl>
<ext_decl>                       ::=    <external_declaration>
                                 ::=    <ext_decl> ; <external_declaration>
```

```
<external_declaration>      ::=   <feature_name> <formal_arguments> <type_mark>
                                  <external_name> <language>
<language>                  ::=   language <string_constant>
<external_name>             ::=
                            ::=   name <string_constant>
<local_variables>           ::=
                            ::=   local <entity_declaration_list>
<body>                      ::=   <full_body>
<full_body>                 ::=   <normal_body>
<normal_body>               ::=   do <compound>
<compound>                  ::=
                            ::=   <comp_list>
<comp_list>                 ::=   <instruction>
                            ::=   <comp_list> ; <instruction>
<instruction>               ::=   <call>
                            ::=   <assignment>
                            ::=   <check>
                            ::=   <retry>
                            ::=   <debug>
<call>                      ::=   <qualified_call>
                            ::=   <unqualified_call>
<qualified_call>            ::=   <expression> . <unqualified_call>
<unqualified_call>          ::=   <feature_name> <actuals>
<actuals>                   ::=
                            ::=   ( <expression_list> )
<expression_list>           ::=
                            ::=   <exp_list>
<exp_list>                  ::=   <expression>
                            ::=   <exp_list> <separator> <expression>
<separator>                 ::=   ,
                            ::=   ;
<assignment>                ::=   <entity> := <expression>
<entity>                    ::=   identifier
                            ::=   Result
<expression>                ::=
                            ::=   <unq_exp_list>
<unq_exp_list>              ::=   <unqualified_expression>
                            ::=   <unq_exp_list> . <unqualified_expression>
<unqualified_expression>    ::=   <constant>
                            ::=   <entity>
                            ::=   <unqualified_call>
                            ::=   Current
                            ::=   <operator_expression>
<operator_expression>       ::=   <unary_expression>
                            ::=   <binary_expression>
                            ::=   <multiary_expression>
                            ::=   <parenthesized>
<unary_expression>          ::=   <unary> <expression>
<unary>                     ::=   not
                            ::=   +
                            ::=   -
<binary_expression>         ::=   <expression> <binary> <expression>
<binary>                    ::=   ^
                            ::=   =
                            ::=   /=
                            ::=   <
                            ::=   >
                            ::=   <=
                            ::=   >=
                            ::=   div
                            ::=   mod
<multiary_expression>       ::=   <expression>
                            ::=   <multiary_expression> <multiary> <expression>
```

```
<multiary>                  ::=   +
                            ::=   -
                            ::=   *
                            ::=   /
                            ::=   and
                            ::=   and then
                            ::=   or
                            ::=   or else
<parenthesized>             ::=   ( <expression> )
<check>                     ::=   check <assertion> end
<retry>                     ::=   retry
<debug>                     ::=   debug <compound> end
<postcondition>             ::=
                            ::=   ensure <assertion>
<rescue>                    ::=
<class_invariant>   ::=
```

e.227

```
<class_declaration>         ::=   <class_header> <formal_generics> <exports> <parents>
                                  <features> <class_invariant> end
<class_header>              ::=   <deferred_mark> class <class_name>
<deferred_mark>             ::=
                            ::=   deferred
<class_name>                ::=   identifier
<formal_generics>           ::=
                            ::=   [ <formal_generic_list> ]
<formal_generic_list>       ::=
                            ::=   <f_gen_list>
<f_gen_list>                ::=   <formal_generic>
                            ::=   <f_gen_list> , <formal_generic>
<formal_generic>            ::=   <formal_generic_name> <constraint>
<formal_generic_name>       ::=   identifier
<constraint>                ::=
                            ::=   -> <class_type>
<exports>                   ::=
                            ::=   export <export_list>
<export_list>               ::=
                            ::=   <ex_list>
<ex_list>                   ::=   <export_item>
                            ::=   <ex_list> , <export_item>
<export_item>               ::=   <feature_name> <export_restriction>
<feature_name>              ::=   identifier
<export_restriction>        ::=
                            ::=   { <class_list> }
<class_list>                ::=
                            ::=   <c_list>
<c_list>                    ::=   <class_name>
                            ::=   <c_list> , <class_name>
<parents>                   ::=
                            ::=   inherit <parent_list>
<parent_list>               ::=
                            ::=   <p_list>
<p_list>                    ::=   <parent>
                            ::=   <p_list> ; <parent>
<parent>                    ::=   <class_type> <rename_clause> <redefine_clause>
<class_type>                ::=   <class_name> <actual_generics>
<actual_generics>           ::=
                            ::=   [ <type_list> ]
<type_list>                 ::=
                            ::=   <t_list>
<t_list>                    ::=   <type>
                            ::=   <t_list> , <type>
```

```
<type>                          ::=   INTEGER
                                ::=   BOOLEAN
                                ::=   CHARACTER
                                ::=   REAL
                                ::=   <class_type>
                                ::=   <formal_generic_name>
                                ::=   <assosciation>
<association>                   ::=   like <anchor>
<anchor>                        ::=   <feature_name>
                                ::=   Current

<rename_clause>                 ::=
                                ::=   <rename <rename_list>

<rename_list>                   ::=
                                ::=   <ren_list>

<ren_list>                      ::=   <rename_pair>
                                ::=   <ren_list> , <rename_pair>

<rename_pair>                   ::=   <feature_name> as <feature_name>
<redefine_clause>               :=
                                ::=   redefine <feature_list>

<feature_list>                  ::=
                                ::=   <feat_list>

<feat_list>                     ::=   <feature_name>
                                ::=   <feat_list> , <feature_name>

<features>                      ::=
                                ::=   feature <feature_declaration_list>

<feature_declaration_list>      ::=
                                ::=   <feat_decl>

<feat_decl>                     ::=   <feature_declaration>
                                ::=   <feat_decl> ; <feature_declaration>

<feature_declaration>           ::=   <feature_name> <formal_arguments> <type_mark> <feature_value_mark>
<formal_arguments>              ::=
                                ::=   ( <entity_declaration_list> )

<entity_declaration_list>       ::=
                                ::=   <ent_decl_list>

<ent_decl_list>                 ::=   <entity_declaration_group>
                                ::=   <ent_decl_list> ; <entity_declaration_group>

<entity_declaration_group>      ::=   <id_list> : <type>
<id_list>                       ::=   identifier
                                ::=   <id_list> , identifier

<type_mark>                     ::=   : <type>
                                ::=

<feature_value_mark>            ::=
                                ::=   is <feature_value>

<feature_value>                 ::=   <constant>
                                ::=   <routine>

<constant>                      ::=   <integer_constant>
                                ::=   <character_constant>
                                ::=   <boolean_constant>
                                ::=   <real_constant>
                                ::=   <string_constant>
<integer_consant>               ::=   <sign> integer
<sign>                          ::=
                                ::=   +
                                ::=   -

<character_constant>            ::=   ' character '
<boolean_constant>              ::=   true
                                ::=   false

<real_constant>                 ::=   <sign> real
<string_constant>               ::=   " string "
<routine>                       ::=   <precondition> <externals> <local_variables> <body>
                                      <postcondtion> <rescue> end

<precondition>                  ::=
                                ::=   require <assertion>
```

```
<assertion>                      ::=
                                 ::=    <assert_list>
<assert_list>                    ::=    <assertion_clause>
                                 ::=    <assert_list> ; <assertion_clause>
<assertion_clause>               ::=    <tag_mark> <unlabeled_assertion_clause>
<tag_mark>                       ::=
                                 ::=    <tag> :
<tag>                            ::=    identifier
<unlabeled_assertion_clause>     ::=    <boolean_expression>
                                 ::=    <comment>
<boolean_expression>             ::=    <expression>
<comment>                        ::=    – string
<externals>                      ::=
                                 ::=    <ext_decl>
<ext_decl>                       ::=    <external_declaration>
                                 ::=    <ext_decl> ; <external_declaration>
<external_declaration>           ::=    <feature_name> <formal_arguments> <type_mark>
                                        <external_name> <language>
<language>                       ::=    language <string_constant>
<external_name>                  ::=
                                 ::=    name <string_constant>
<local_variables>                ::=
                                 ::=    local <entity_declaration_list>
<body>                           ::=    <full_body>
<full_body>                      ::=    <normal_body>
<normal_body>                    ::=    do <compound>
<compound>                       ::=
                                 ::=    <comp_list>
<comp_list>                      ::=    <instruction>
                                 ::=    <comp_list> ; <instruction>
<instruction>                    ::=    <call
                                 ::=    <assignment>
                                 ::=    <conditional>
                                 ::=    <loop>
                                 ::=    <check>
                                 ::=    <retry>
                                 ::=    <debug>
<call>                           ::=    <qualified_call>
                                 ::=    <unqualified_call>
<qualified_call>                 ::=    <expression> . <unqualified_call>
<unqualified_call>               ::=    <featire_name> <actuals>
<actuals>                        ::=
                                 ::=    ( <expression_list> )
<expression_list>                ::=
                                 ::=    <exp_list>
<exp_list>                       ::=    <expression>
                                 ::=    <exp_list> <separator> <expression>
<separator>                      ::=    ,
                                 ::=    ;
<assignment>                     ::=    <entity> := <expression>
<entity>                         ::=    identifier
                                 ::=    Result
<expression>                     ::=
                                 ::=    <unq_exp_list>
<unq_exp_list>                   ::=    <unqualified_expression>
                                 ::=    <unq_exp_list> . <unqualified_expression>
<unqualified_expression>         ::=    <constant>
                                 ::=    <entity>
                                 ::=    <unqualified_call>
                                 ::=    Current
                                 ::=    <old_value>
                                 ::=    <no_change>
                                 ::=    <operator_expression>
```

```
<old_value>                 ::=    old <expression>
<nochange>                  ::=    nochange
<operator_expression>       ::=    <unary_expression>
                            ::=    <binary_expression>
                            ::=    <multiary_expression>
                            ::=    <parenthesized>
<unary_expression>          ::=    <unary> <expression>
<unary>                     ::=    not
                            ::=    +
                            ::=    -
<binary_expression>         ::=    <expression> <binary> <expression>
<binary>                    ::=    ^
                            ::=    =
                            ::=    /=
                            ::=    <
                            ::=    >
                            ::=    <=
                            ::=    >=
                            ::=    div
                            ::=    mod
<multiary_expression>       ::=    <expression>
                            ::=    <multiary_expression> <multiary> <expression> <multiary>
                            ::=    +
                            ::=    -
                            ::=    *
                            ::=    /
                            ::=    and
                            ::=    and then
                            ::=    or
                            ::=    or else
<parenthesized>             ::=    ( <expression> )
<conditional>               ::=    if <then_part_list> <else_part> end
<then_part_list>            ::=    <then_part>
                            ::=    <then_part_list> elsif <then_part>
<then_part>                 ::=    <boolean_expression> then <compound>
<else_part>                 ::=
                            ::=    else <compound>
<loop>                      ::=    <initialization> <loop_invariant> <loop_variant> <exit_clause>
                                   <loop_body> end
<initialization>            ::=    from <compound>
<loop_invariant>            ::=
                            ::=    invariant <assertion>
<loop_variant>              ::=
                            ::=    variant <tag_mark> <integer_expression>
<integer_expression>        ::=    <expression>
<exit_clause>               ::=    until <boolean_expression>
<loop_body>                 ::=    loop <compound>
<check>                     ::=    check <assertion> end
<retry>                     ::=    retry
<debug>                     ::=    debug <compound> end
<postcondition>             ::=
                            ::=    ensure <assertion>
<rescue>                    ::=
                            ::=    rescure <compound>
<class_invariant>           ::=
                            ::=    invariant <assertion>
```

The following grammars describe computer commands.

i.07

```
<clist>          ::=
                 ::=      <command> <clist>
<command>        ::=      <trans>
                 ::=      <lrsys>
<lrsys>          ::=      lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>          ::=      trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                 ::=      trans i=(basistok, cray), o=(basistok, cray, noc)
```

i.09

```
<clist>          ::=
                 ::=      <command> <clist>
<command>        ::=      exe  mppl config <file>
                 ::=      exe  mppl mac <file>
                 ::=      mppl
                 ::=      civic i= <file> , o=g
                 ::=      ldr i= <file> , lib = basislib, x= <file>
                 ::=      lib mppl x. basislib
file>            ::=      string
```

i.12

```
<clist>          ::=
                 ::=      <command> <clist>
<command>        ::=      <compile>
                 ::=      <build>
                 ::=      <trans>
                 ::=      <lrsys>
<file>           ::=      string
<compile>        ::=      rcft i= <file>
<build>          ::=      build nl = <file>, b=<file>, lib=(grafl3,fortlib,nag,slatec)
<lrsys>          ::=      lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>          ::=      trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                 ::=      trans i=(basistok, cray), o=(basistok, cray, noc)
```

i.17

```
<clist>          ::=
                 ::=      <command> <clist>
<command>        ::=      exe  mppl config <file>
                 ::=      exe  mppl mac <file>
                 ::=      mppl <files> <mout>
                 ::=      civic i= <file> , o=g
                 ::=      ldr i= <file> , lib = basislib, x= <file>
                 ::=      lib mppl x. basislib
<files>          ::=
                 ::=      <filelist>
<filelist>       ::=      <file>
                 ::=      <file> <separator> <file>
<separator>      ::=
                 ::=      ,
<mout>           ::=
                 ::=      > <file>
<file>           ::=      string
```

i.18

```
<clist>          ::=
                 ::=      <command> <clist>
```

```
<command>          ::=     <mppl>
                   ::=     <config>
                   ::=     <mac>
                   ::=     <compile>
                   ::=     <build>
                   ::=     <trans>
                   ::=     <lrsys>
<mppl>             ::=     mppl
<config>           ::=     exe  mppl config <file>
<mac>              ::=     exe  mppl mac <file>
<file>             ::=     string
<compile>          ::=     rcft i= <file>
<build>            ::=     build nl = <file>, b=<file>, lib=(grafl3,fortlib,nag,slatec)
<lrsys>            ::=     lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>            ::=     trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                   ::=     trans i=(basistok, cray), o=(basistok, cray, noc)
```

<u>i.26</u>

```
<clist>            ::=
                   ::=     <command>  <clist>
<command>          ::=     <config>
                   ::=     <mac>
                   ::=     <mppl>
                   ::=     <compile>
                   ::=     <build>
                   ::=     <trans>
                   ::=     <lrsys>
<config>           ::=     exe  mppl config <file>
<mac>              ::=     exe  mppl mac <file>
<mppl>             ::=      mppl <files> <mout>
<files>            ::=
                   ::=     <filelist>
<filelist>         ::=     <file>
                   ::=     <file> <separator> <filelist>
<mout>             ::=     > <file>
                   ::=
<separator>        ::=
                   ::=     ,
<file>             ::=     string
<compile>          ::=     rcft i= <file>
<build>            ::=     build nl = <file>, b=<file>, lib=(grafl3,fortlib,nag,slatec)
<lrsys>            ::=     lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>            ::=     trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                   ::=     trans i=(basistok, cray), o=(basistok, cray, noc)
```

<u>i.30</u>

```
<clist>            ::=
                   ::=     <command>  <clist>
<command>          ::=     <config>
                   ::=     <mac>
                   ::=     <mppl>
                   ::=     <compile>
                   ::=     <load>
                   ::=     <build>
                   ::=     <trans>
                   ::=     <lrsys>
<config>           ::=     exe  mppl config <file>
<mac>              ::=     exe  mppl mac <file>
<mppl>             ::=      mppl <files> <mout>
<files>            ::=
                   ::=     <filelist>
```

```
<filelist>          ::=     <file>
                    ::=     <file> <separator> <filelist>
<mout>              ::=     > <file>
                    ::=
<separator>         ::=
                    ::=     ,
<file>              ::=     string
<compile>           ::=     civic i= <file> , o=g
                    ::=     rcft i= <file>
<build>             ::=     build ol= <file> , b= <file>
                    ::=     build nl = <file>, b=<file>, lib=(grafl3,fortlib,nag,slatec)
<load>              ::=     ldr i= <file>, x=<file>
<lrsys>             ::=     lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>             ::=     trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                    ::=     trans i=(basistok, cray), o=(basistok, cray, noc)
```

i.33

```
<clist>             ::=
                    ::=     <command>  <clist>
<command>           ::=     <config>
                    ::=     <mac>
                    ::=     <mppl>
                    ::=     <compile>
                    ::=     <load>
                    ::=     <build>
                    ::=     <trans>
                    ::=     <lrsys>
                    ::=     <extract>
<config>            ::=     exe  mppl config <file>
<mac>               ::=     exe  mppl mac <file>
<mppl>              ::=      mppl <files> <mout>
<files>             ::=
                    ::=     <filelist>
<filelist>          ::=     <file>
                    ::=     <file> <separator> <filelist>
<mout>              ::=     > <file>
                    ::=
<separator>         ::=
                    ::=     ,
<file>              ::=     string
<compile>           ::=     civic i= <file> , o=g
                    ::=     rcft i= <file>
<build>             ::=     build ol= <file> , b= <file>
                    ::=     build nl = <file>, b=<file>, lib=(grafl3,fortlib,nag,slatec)
<load>              ::=     ldr i= <file> , lib = basislib, x= <file>
                    ::=     ldr i= <file>, x=<file>
<lrsys>             ::=     lrsys bnf=basisbnf, tok=basistok, fortran77= <file>
<trans>             ::=     trans i=(basisbnf, cray), o=(basisbnf, cray, noc)
                    ::=     trans i=(basistok, cray), o=(basistok, cray, noc)
<extract>           ::=     lib mppl x. basislib
```

The following grammars describe subsets of the Basis utility Config.

c.06

```
<config_file>       ::=     <header>  <name> <title> <iter>
<header>            ::=     package
                    ::=     foreign
```

```
<name>              ::=     string
<title>             ::=     string
<iter>              ::=     integer
```

c.15

```
<config_file>       ::=     <header> <name> <title> <iter> <rootlist>
<header>            ::=     package
                    ::=     foreign
<name>              ::=     string
<title>             ::=     string
<iter>              ::=     integer
<rootlist>          ::=
                    ::=     <root> <rootlist>
<root>              ::=     init
                    ::=     gen
                    ::=     genp
                    ::=     exe
                    ::=     exep
                    ::=     fin
                    ::=     finp
```

c.21

```
<config_file>       ::=     <header> <name> <title> <iter> <rootlist> <opts>
<header>            ::=     package
                    ::=     foreign
<name>              ::=     string
<title>             ::=     string
<iter>              ::=     integer
<rootlist>  ::=
                    ::=     <root> <rootlist>
<root>              ::=     init
                    ::=     gen
                    ::=     genp
                    ::=     exe
                    ::=     exep
                    ::=     fin
                    ::=     finp
<opts>              ::=
                    ::=     <option> <opts>
<option>            ::=     codename string
                    ::=     codefile string
                    ::=     cprompt string
                    ::=     lcprompt integer
```

c.29

```
<config_file>       ::=     <header> <name> <title> <iter> <rootlist> <opts>
<header>            ::=     package
                    ::=     foreign
<name>              ::=     string
<title>             ::=     string
<iter>              ::=     integer
<rootlist>          ::=
                    ::=     <root> <rootlist>
<root>              ::=     init
                    ::=     gen
                    ::=     genp
                    ::=     exe
                    ::=     exep
                    ::=     fin
                    ::=     finp
```

```
<opts>          ::=
                ::=     <option> <opts>
<option>        ::=     codename string
                ::=     codefile string
                ::=     cprompt string
                ::=     lcprompt integer
                ::=     firstpkg string
                ::=     macfile string
                ::=     probname string
                ::=     userbox box string
                ::=     verbose <answer>
                ::=     echo <answer>
<answer>        ::=     yes
                ::=     no
```

APPENDIX D -- Input Strings

Appendix D contains the input strings used in the timings of the second part of the recognizer, REC2.

Basis Input Strings

b.a.3

```
integer name ;
```

b.a.7

```
integer name ;
name = name ;
```

b.a.14

```
integer name ;
name = name ;
integer name ;
name = name ;
```

b.a.28

```
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
```

b.a.56

```
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
name = name ;
integer name ;
```

b.b.9

```
integer name ;
name = name + name ;
```

b.b.18

```
integer name ;
name = name + name ;
integer name ;
name = name + name ;
```

b.b.36

```
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
```

b.b.72

```
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
integer name ;
name = name + name ;
```

b.c.11

```
integer name , name ;
name = name + name ;
```

b.c.22

```
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
```

b.c.44

```
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
```

b.c.88

```
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
integer name , name ;
name = name + name ;
```

b.d.17

```
integer name , name ;
name = name + name ;
name = name * name ;
```

b.d.34

```
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
```

b.d.68

```
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
```

b.d.136

```
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
integer name , name ;
name = name + name ;
name = name * name ;
```

b.e.1

```
name
```

b.e.3

```
name + name
```

b.e.7

```
name + name + name + name
```

b.e.15

```
name + name +  name + name + name + name + name + name
```

b.f.5

```
name + name * name
```

b.f.7

```
name + name * name ** name
```

b.f.9

```
( name + name * name ** name )
```

b.f.11

```
( ( name + name * name ** name ) )
```

Eiffel Input Strings

e.a.3

```
class identifier
end
```

e.a.7

```
class identifier
feature identifier : INTEGER
end
```

e.a.15

```
class identifier
feature identifier : INTEGER ;
            identifier : REAL ;
            identifier : INTEGER
end
```

e.a.31

```
class identifier
feature identifier : INTEGER ;
            identifier : REAL ;
            identifier : INTEGER ;
            identifier : REAL ;
            identifier : INTEGER ;
            identifier : REAL ;
            identifier : INTEGER
end
```

e.b.15

```
class identifier
export identifier
inherit identifier
feature identifier : INTEGER ;
            identifier is true
end
```

e.b.30

```
class identifier
export identifier , identifier
inherit identifier ; identifier
feature identifier : INTEGER ;
            identifier : REAL ;
            identifier is " string " ;
            identifier is - real
end
```

e.b.45

```
class identifier
export identifier , identifier , identifer ,
            identifier , identifier , identifier
inherit identifier ; identifier
feature identifier : INTEGER ;
            identifier : REAL ;
            identifier is true ;
            identifier is real ;
            identifier is " string " ;
            identifier is false
end
```

e.b.60

```
class identifier
export identifier , identifier , identifer ,
            identifier , identifier , identifier ,
            identifier , identifier , identifier , identifier
inherit identifier ; identifier
feature identifier : INTEGER ;
            identifier : REAL ;
            identifier is true ;
            identifier is  - real ;
            identifier is " string " ;
            identifier is false ;
            identifier is ' character '
end
```

Command Input Strings

i.a.13

```
lib mppl x. basislib
exe  mppl config string
exe  mppl mac string
mppl
```

i.a.25

```
lib mppl x. basislib
exe  mppl config string
exe  mppl mac string
mppl
civic i= string , lib=basislib, x= string
```

i.b.11

```
trans i=(basisbnf,cray), o=(basisbnf,cray,noc)
trans i=(basistok,cray), o=(basistok,cray,noc)
lrsys bnf=basisbnf, tok=basistok, fortran77= string
```

i.b.22

```
trans i=(basisbnf,cray), o=(basisbnf,cray,noc)
trans i=(basistok,cray), o=(basistok,cray,noc)
lrsys bnf=basisbnf, tok=basistok, fortran77= string
rcft i= string
build nl=basislib, b= string , lib=(grafl3,fortlib,nag,slatec)
```

i.b.31

```
trans i=(basisbnf,cray), o=(basisbnf,cray,noc)
trans i=(basistok,cray), o=(basistok,cray,noc)
lrsys bnf=basisbnf, tok=basistok, fortran77= string
rcft i= string
build nl=basislib, b= string , lib=(grafl3,fortlib,nag,slatec)
exe  mppl config string
exe  mppl mac string
mppl
```

i.b.34

```
trans i=(basisbnf,cray), o=(basisbnf,cray,noc)
trans i=(basistok,cray), o=(basistok,cray,noc)
lrsys bnf=basisbnf, tok=basistok, fortran77= string
rcft i= string
build nl=basislib, b= string , lib=(grafl3,fortlib,nag,slatec)
exe  mppl config string
exe  mppl mac string
mppl string > string
```

<u>i.b.48</u>

```
trans i=(basisbnf,cray), o=(basisbnf,cray,noc)
trans i=(basistok,cray), o=(basistok,cray,noc)
lrsys bnf=basisbnf, tok=basistok, fortran77= string
rcft i= string
build nl=basislib, b= string , lib=(grafl3,fortlib,nag,slatec)
exe  mppl config string
exe  mppl mac string
mppl string > string
civic i= string , o=g
build ol= string , b= string
ldr i= string , x= string
```

Config Input Strings

<u>c.a.04</u>

```
package
string
string
integer
```

<u>c.b.05</u>

```
package
string
string
integer
init
```

<u>c.b.11</u>

```
package
string
string
integer
init  gen  genp  exe  exep  fin  finp
```

<u>c.c.07</u>

```
package
string
string
integer
init

codename string
```